Diplomarbeit

im Studiengang Medieninformatik

an der Fachhochschule Stuttgart, Hochschule der Medien

Standardisierte Integration Mobiler Endgeräte an eine universelle Kommunikationsinfrastruktur für die Produktion

Betreuung:

Prof. Walter Kriha

Dipl. Inf. Gerald Knoll

Vorgelegt am 1. September 2004 von:

Ralf Tischer

Fraunhofer Institut Produktionstechnik und Automatisierung

FACHHOCHSCHULE STUTTGART
HOCHSCHULE DER MEDIEN

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Stuttgart, Mittwoch, 1. September 2004

Ralf Tischer

Abstract

The goal of this diploma thesis was the realization of a standard web access to a communication framework (Message to Anywhere) used by highly automated semiconductor factories. Mobile Devices, such as Mobile phone, PDA or Tablet PC will be able to connect to Message to Anywhere via a web server.
Message to Anywhere allows an easy integration of equipments into the framework. These equipments can make their functionality available in the framework as services.
Therefore a concept was developed that is able to map these services upon different mobile devices to allow remote control the equipments.
The data delivered by the framework must be customized to the different requesting devices.
As a concrete example the services of a 3D purification plant will be made available upon different mobile devices.

Kurzfassung

Ziel dieser Diplomarbeit war die Realisierung eines standardisierenden webbasierten Zugangs für ein Kommunikationsframework (Message to Anywhere) zu schaffe. Dieses Framework wird in hoch automatisierten Halbleiterfabriken zum Einsatz kommen.
Mobile Endgeräte wie Mobiltelefonen, PDAs oder Tablet PCs könne sind über einen Webserver mit Message to Anywhere zu verbinden.
Das Framework erlaubt eine einfache Integration von Anlagen. Diese stellen ihre Funktionen in Form von Services innerhalb des Frameworks bereit. Hierfür wurde ein Konzept entwickelt, dass es erlaubt diese Services auf verschieden Mobilen Endgeräten zur Analagensteuerung abzubilden. Die Daten, die das Framework hierbei liefert, müssen auf die unterschiedlichen Endgeräte angepasst werden.
Als eine Beispielanwendung für die Analagensteuerung werden die Services einer
3D- Reinigungsanlage auf unterschiedlichen Mobilen Endgeräten zu Verfügung gestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Zielsetzung und Problemstellung

Der Einsatz moderner Software-Technologien und neuer Mobiler Endgeräte (PDA, Smartphone, TabletPC,...) bietet hervorragende Möglichkeiten einer universellen und mobilen Anlagensteuerung. Im Rahmen des Forschungsprojekt Message to Anywhere (M2A) wird eine Kommunikationsinfrastruktur entwickelt, die in der Hightech-Branche eingesetzt werden soll. Durch Mobile Endgeräte können hierüber Anlagen aus der Halbleiterfertigung angesteuert werden.

Ein Ziel dieser Arbeit ist es, einen standardisierten Zugang zum Message to Anywhere Framework über einen Webserver für verschiedene Mobile Endgeräte zu schaffen. Zur flexiblen Integration von Anlagen und Applikationen in das Framework muss weiterhin ein Konzept entwickelt werden, das Services (Dienste) dieser Anlagen auf verschieden Mobilen Endgeräten zur Verfügung stellt. Die zur Verfügung stehenden Informationen müssen an verschiedenen Ausgabekanälen und an die sehr unterschiedlichen Darstellungsmöglichkeiten Mobiler Endgeräte adaptiert werden. Das entwickeltes Konzept wird als Prototyp implementiert und in das Message to Anywhere Framework integriert.

Als konkrete Beispielanwendung einer Analagensteuerung werden die Funktionen einer 3D-Reinigungsanlage (teilweise) auf verschiedenen Mobilen Endgeräten zugänglich gemacht.

Die Herausforderungen liegen auf der einen Seite ein flexibles Konzept zu entwerfen, das es erlaubt die unterschiedlichen Funktionen einer Maschine abzubilden, und in das Message to Anywhere Framework zu integreren. Die zweite große Herausforderung ist es, diese Struktur über einen Webserver auf verschiedenen Endgeräten und Ausgabekanälen zur Verfügung zu stellen und so den sehr unterschiedlichen Geräten einen standardisierten Zugriff zu ermöglichen. Dadurch muss auf den Mobilen Geräten keinen zusätzliche Software entwickelt bzw. installiert werden.

Da sich Message to Anywhere noch in der Entwicklung befindet, muss der Webserver zukunftssicher und flexibel aufgebaut werden, um die Integration weiterer Anwendungen zu ermöglichen.

1.2 Aufbau der Diplomarbeit

Kapitel 2: Dieses Kapitel beschäftigt sich mit der Analyse der Aufgabenstellung. Am Anfang wird hier eine (kurze) Übersicht über die aktuellen Gerätegruppen Mobiler Endgeräte gegeben. Im zweiten Abschnitt werden verschieden Webframeworks beleuchtet, die für die Verwirklichung der Aufgabenstellung im praktischen Teil der Arbeit in Frage kamen.

Kapitel 3: Hier werden Technologien vorgestellt, die zum Einsatz kamen. Ein Schwerpunkt ist hierbei das Webframework Cocoon, auf dem der Webserver im Praktische Teil der Arbeit aufbaut.

Kapitel 4: In diesem Kapitel wird das Kommunikationsframework Message to Anywhere vorgestellt. Das Service-Konzept von Message to Anywhere steht hier im Vordergrund, da dies eine wichtige Rolle in dieser Arbeit gespielt hat.

Kapitel 5: Dieses Kapitel widmet sich der Verwirklichung des Prototypen und dessen Integration an das Message to Anywhere Framework. Zu Beginn wird die Konzeption und Planung für den Webserver vorgestellt. Hieran anknüpfend werden die einzelnen Pakete, in die die Implementation aufgeteilt ist, ausführlich besprochen.

2 Analyse

Die Aufgabenstellung kann grundsätzlich als Client- Server Szenario betrachtet werden. Bei der Analyse wurde zunächst Clientseitig (Kapitel 2.1) eine Recherche über die unterschiedlichen Mobile Endgeräte auf dem Markt durchgeführt und welche verschiedenen Eingabeformate diese verarbeiten können.
Serverseitig (Kapitel 2.2) wurden zunächst die Anforderungen herausgearbeitet die M2A an einen Webserver stellt. Anhand dessen wurden nach einer Vorauswahl drei Webframeworks ausgewählt, und daraufhin untersucht wie gut sie die gestellten Anforderungen erfüllen.

2.1 Geräteklassen von Mobiler Endgeräte

Dieser Überblick über Geräteklassen Mobiler Endgeräte ist keine Vorstellung aktueller Produkte, da diese bei der rasanten Entwicklung auf dem Markt in kürzester Zeit überholt sein würden. Vielmehr werden die Gerätegruppen, die sich mit der Zeit herauskristallisiert haben, vorstellt.
Ein Augenmerk lag hierbei auf den Verschiedenen Formaten die solche Geräte verarbeiten können und entsprechend vom Server geliefert werden müssen. Die Ausgabeformate Hypertext Markup Language (HTML), Wireless Application Protocol (WML) und Portable Document Format (PDF) Format wurde hierzu ausgewählt.

Mobiltelefon

Die Einführung der Mobiltelefone, oder auch kurz Handy genannt, macht die Möglichkeit der mobilen Kommunikation für jedermann zugänglich. Ältere Geräte waren noch stark eingeschränkt in der Darstellung von Informationen, und primär zum Telefonieren gedacht. Mobiltelefone der neueren Generation bieten durch hochauflösende Farbdisplays und entsprechender Browsersoftware neue Möglichkeiten.

Im Bereich der Mobiltelefone gibt es kein einheitliches Betriebssystem, praktisch jeder Hersteller entwickelt hier seine eigene Lösung. Die Darstellung von Informationen auf den verschiedenen Geräten ist schwierig, da es keine genormten Bildschirmgrößen gibt. Das einzige unterstützte Ausgabeformat ist WML.

Smartphone

Der wohl grundsätzliche Unterschied, der ein Handy von einem Smartphone unterscheidet, stellt das Betriebssystem da. Zurzeit teilen sich im Wesentlichen drei Hersteller den Markt für solche Betriebssysteme. Die Lösungen der Firmen PALM und Microsoft scheinen sich zurzeit nicht auf dem Markt durchzusetzen. Das Betriebssystem Symbian ist Marktführer in diesem Bereich. Es wurde speziell für den Einsatz auf Mobilen Endgeräten entwickelt und kann sehr flexibel auf die verschiedenen Anforderungen der Smartphones regieren. Eine klare Abgrenzung zwischen einem PDA und einem Smartphone zu zeichnen wird immer schwieriger, da aktuelle Smartphones immer mehr PDA typische Funktionalitäten adaptieren und es voraussichtlich mittelfristig zu einer Verschmelzung der Gerätegruppen kommen wird. Es sind aktuell Smartphones auf dem Markt, die ein großes Display mit Stifteingabe, ausreichend Speicher und eine schnelle CPU besitzen, so dass das Bearbeiten von Präsentationen oder Textdokumenten kein Problem mehr darstellt. [Gerlicher04]
Im Wesentlichen wird von dieser Gerätegruppe die Ausgabeformate WML und HTML unterstützt. Auch Software zum Verarbeiten von PDF ist vorhanden, allerdings sind hierfür die Bildschirmabmessungen etwas klein.

Personal Digital Assistent (PDA)

Die PDAs haben sich von einem elektronischen Terminkalender und Adressbuch zu einem echten Minicomputer entwickelt. Sie verfügen über genügend Rechenleistung, um beispielsweise ein Textverarbeitungsprogramm oder Multimediaanwendungen auszuführen. PDAs besitzen meist keine Tastatur, sondern werden mittels berührungsempfindlichen Display bedient.
Für PDAs steht eine große Anzahl Software zur Verfügung, die kaum Wünsche offen lässt. Der Markt wird größtenteils von zwei Betriebssystemen beherrscht: Zum einem gibt es hier die Palm OS betriebenen Geräte, die auch vom Hersteller mit eigener Hardware vertrieben werden. Daneben existieren noch Geräte die mit dem Betriebssystem Pocket PC (ehemals Windows CE) von Microsoft ausgerüstet sind. Auch eine Linux Distribution ist für PDAs

erhältlich. Ein Mobile Digital Assistant (MDA) stellt eine Verschmelzung zwischen einem Handy und einem PDA da. Teilweise wird ein MDA auch unter die Gerätegruppe der Smartphone eingeordnet. Die Unterstützten Ausgabeformate sind HTML und PDF.

TabletPC

Tablet PCs ähneln sehr der Geräteklasse des Notebooks. Sie sind allerdings deutlich kleiner und leichter und verfügen über ein dreh- und klappbares Display, dass das Arbeiten mit einem Stift erlaubt.

Sie besitzen auch eine Tastatur, und über die gängigen Schnittstellen eines Desktop PC, so dass sich jedes beliebige Eingabe- und Anzeigegerät anschließen lässt. Die Prozessorleistung ist geringer wie bei Notebooks und die Geräte kommen häufig ohne eine aktive Kühlung aus. Praktisch alle Hersteller verwenden als Betriebssystem Windows XP Tablet PC Edition. Diese Geräteklasse unterstützt alle Ausgabeformate.

Notebook

Heutige Notebooks stehen einem Desktop PC in Leistung und Ausstattung in nichts mehr nach. Alle gängigen Ein- und Ausgabegeräte können problemlos angeschlossen werden. Bei Notebooks gibt es enorme Unterschiede in Größe, Gewicht und Ausstattung. Dies führt zu einem sehr breiten Einsatzspektrum solcher Geräte in der Mobilen Kommunikation. Auf allen Geräten können praktisch alle gängigen Betriebssysteme eingesetzt werden. Ebenso wie bei der Geräteklasse der Tablet PCs gibt es hier keinen Einschränkung bezüglich des Ausgabeformats.

Tabelle 1 zeigt noch einmal eine Gegenüberstellung aller hier beschrieben Geräteklassen. Grundsätzlich sind alle Geräte für die mobile Anlagensteuerung geeignet. Für eine sinnvolle und handhabbare Anwendung müssen die Informationen auf die jeweiligen Gegebenheiten des Endgerätes angepasst werde. Insbesondere bei der Größe des Displays sind die Unterschiede enorm.

	Mobiltelefon	Smartphone	PDA	Tablet PC	Notebook
Auflösung	84x84 101x80 160x128 und weitere	220x176 320x208 320x240	320x204 selten weitere Formate	800x600 1024x768	800x600 1024x768 1280x1024
Speicher/ MB	1-4	3,4 - 32	16-128	256 -512 und mehr	256 -1024 und mehr
Unterstütze Ausgabeformate	WML	HTML/ WML bedingt auch PDF	HTML/PDF	keine Einschränk.	keine Einschränk.
Eingabe-möglichkeiten	Tel. Tastatur/ Steuerkreuz	Tel. Tastatur/Stift Steuerkreuz	Stift / selten Tastatur	Tastatur/ Maus / Stift	Tastatur/ Maus
Betriebssystem	Hersteller spezifisches Betriebssystem	Symbian OS	Windows CE / emb. Linux/ PalmOS	Windows XP Tablet PC Edition	Windows / Linux

Tabelle 1: Geräteklassen im Überblick [CTspecial04]

2.2 Auswahl der Serverseitigen Technologie

Einige Rahmenbedingungen, die das Message to Anywhere Projekt vorgab waren bei der Auswahl der Serverseitigen Technologie zu berücksichtigen:

XML-Unterstützung: Das Framework wird die Daten in Form von Extensible Markup Language (XML) liefern. Der aktuelle Prototyp unterstützt dies noch nicht, allerdings wird es in einem nächsten Release verwirklicht und muss berücksichtigt werden.

Open Source: Da es sich bei dem gesamten M2A-Projekt um ein Open Source Projekt handelt, müssen alle verwendeten Komponenten ebenfalls Open Source sein.

Qualität: Es muss ein ausgereiftes und stabiles Webframework als Basis verwendet werden, um den Qualitätsansprüchen von M2A zu genügen. Des weitern muss die gewählte Lösung zukunftssicher sein.

Ausgabeformat: Die Informationen sollen in verschiedenen Ausgabeformate (HTML, WML, PDF,...) zur Verfügung gestellt werden, abhängig von dem anfragenden Mobilen Endgerät.

Ausgabegeräte: Die Informationen müssen auf unterschiedlichen Mobilen Endgeräten abrufbar sein. Abhängig von den Eigenschaften des Gerätes muss die Darstellung der Ausgabe entsprechend angepasst werden.

Flexibilität: Einen entwickelte Lösung soll flexibel auf neue Anforderungen reagieren können und die Integration neuer Anwendungen soll einfach möglich sein.

Der Webbasierte Zugang zum M2A-Framework sollte von Anfang an auf einem Webframework aufbauen. Der Vorteil von solchen Frameworks liegt darin, dass es die Web-Programmierung als solche abstrahiert und so mehr Raum für die Lösung des eigentlichen Problems schafft. Webframeworks bieten beispielsweise Schnittstellen für das Überwachen von Sessions, unterstützen die Trennung von Daten und Layout, erleichtern den Zugriff auf Fremdsysteme oder legen ein einheitliches Vorgehen bei der Entwicklung fest.
Es gibt eine Vielzahl von Webframeworks. Eine komplette Übersicht und eine Vergleichsmatrix der einzelnen Produkte bietet das Wafer Projekt [Wafer04]. Praktisch alle Frameworks sind in Java entwickelt und Open Source, jedoch bauen sie teilweise auf völlig unterschiedlichen Konzepten auf, die zu einer unterschiedlichen Sichtweise auf die Daten und deren Verarbeitung führt.
Nach einer Vorauswahl wurden drei populäre Frameworks, Struts [Struts04], Turbine [Turbine04] und Cocoon [Cocoon04], genauer unter die Lupe genommen in wieweit sie den Anforderungen von M2A genügen.
Hinter allen drei Frameworks steht eine non Profit Organisation, die Apache Software Foundation [Apache04]. Diese ging ursprünglich aus dem Apache HTTP-Server Projekt hervor und wird von fast allen wichtigen Unternehmen der IT-Branche unterstützt.

Struts

Dies ist wohl das zurzeit am weitesten verbreitete und beliebteste Webframework. Es wird oft auch als ein Quasistandard auf diesem Gebiet bezeichnet. Struts verfolgt konsequent den Model View Control (MVC) Ansatz wie in Abbildung 1 beschrieben.

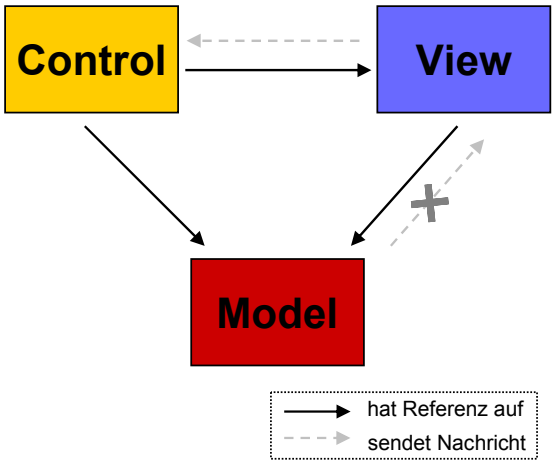


Abbildung 1: Model View Control (MVC) in Struts

Das Model definiert die Datenstruktur der Anwendung. Daten werden im Model gespeichert und können verändert werden. Die View stellt die Daten aus dem Model auf dem Bildschirm da. Ein Benutzer kann auf der View Aktionen ausführen auf die die Control reagiert. Anhand dieser Benutzereingabe werden die Daten im Model verändert. Im klassischen MVC kann das Model die View über neue Daten benachrichtigen. Da Hypertext Transfer Protocol (HTTP) ein Zustandsloses Protokoll ist entfällt diese Benachrichtigung. Zustandslos bedeutet, dass der Server nach jedem Request (Anfrage) wieder in den Grundzustand zurückkehrt, ein nachfolgender Request kann sich nicht unmittelbar auf den Vorherigen beziehen. Struts verwendet den so genannten Modell 2 Ansatz, der eine Übertragung des MVC-Modells entspricht, und für Webanwendungen eine sauberere Umsetzung des MVC-Paradigmas erlaubt.

Struts basiert auf verschiedenen Standardtechnologien. Für die View sind Java Server Pages (JSP) zuständig. Eine Änderung des Views wird von dem Controller, ein so genanntes ActionServlet, entgegengenommen. Der Anwendungsfluss selbst wird in einer Konfigurationsdatei angegeben, die vom ActionServlet benutzt wird. Dieses Servlet verändert die Daten im Model, das als Java Bean verwirklicht wurde. Eine große Stärke von Struts liegt in der Formularbehandlung. Hierunter versteht man die Generierung von Eingabemasken für den Benutzer und die Validierung der eingegebenen Daten. Eine weitere Stärke ist die Integration in bestehende Server-Systeme und die Unterstützung durch verschiedene Entwicklungsumgebungen. Die Generierung verschiedener Ausgabeformate und das Verarbeiten von XML-Daten werden von Struts, im Vergleich zu Cocoon, nicht optimal unterstützt. [Struts04]

Cocoon

In erster Linie ist Cocoon ein XML-Publishingframework. Die Grundfunktionalität ist, abhängig von dem Request eines Clients, aus einem oder mehreren XML-Dokument(en) mit Hilfe entsprechender XSLT-Stylesheets (Extensible Stylesheet Language Transformation) ein beliebiges Ausgabeformat zu erzeugen. Das Ergebnis wird als Response zurückgeliefert.

Cocoon besitzt eine strikte Trennung von Content (Daten), Style (Layout) und Logic (Logik). Dieses Konzept wird als Separation of Concerns (SoC) bezeichnet. Es ermöglicht eine Aufteilung von Projekten in verschiedene Arbeitsbereiche, wie in Abbildung 2 zu sehen ist. Content steht für die redaktionellen, inhaltlichen Aufgaben. Die Integration der dynamischen Inhalte wird von der Logic übernommen. Style übernimmt die Präsentation der Daten, Grafiken und das Look & Feel. Alles wird von einem Management überwacht, das sich um Inhalte, Abläufe und Aufbau kümmert. Bei großen Projekten kann sich eine strikte Einhaltung von SoC bezahlt machen, da es eine saubere Arbeitsteilung ermöglicht.

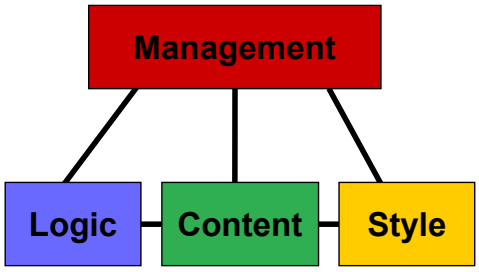


Abbildung 2: Separation of Concerns (SoC) mit Cocoon [Kumar04]

Cocoon wird in der Regel als Servlet in einem Webserver betrieben, kann aber auch alleine arbeiten. Ein Request wird von Cocoon entgegengenommen, analysiert und anhand einer Konfigurationsdatei (Sitemap) werden die nötigen Arbeitschritte (Pipelines) ausgewählt. Innerhalb der Pipelines erfolgt die Verarbeitung sukzessiv. Seine großen Stärken hat Cocoon in der uneingeschränkten Unterstützung von XML und der komponentenbasierten Architektur, die sehr flexibel ist. Die Schwächen liegen zurzeit noch in der Verarbeitung von Formulardaten. Der Aufbau einer komplexen Webanwendung wird durch die Verwendung des Pipelinekonzeptes etwas erschwert, bzw. verlangt eine intensive

Einarbeitung. Leider steht für Cocoon keine echte Entwicklungsumgebung zur Verfügung, sondern man muss auf XML-Entwicklungswerkzeuge zurückgreifen. [Kumar04]

Turbine

Turbine baut, wie auch Struts, auf der MVC-Architektur (Abbildung 1) auf. Es setzt auf einem zentralen Servlet auf. Alle Requests werden zunächst über dieses Servlet verarbeitet. Bei der Verarbeitung geht Turbine immer davon aus, dass eine Seite aus den vier Elementen (man spricht hier auch von Modulen) Page, Layout, Navigation und Screen besteht wie in Abbildung 3, linker Teil ersichtlich. Bei einem Request wird zuerst die Page, das erste Modul in der Verarbeitungskette (Abbildung 3, rechter Teil), ausgeführt. Die Verarbeitungslogik, beispielsweise das Validieren einer Formulareingabe, wird in so genannten Action Klassen implementiert.

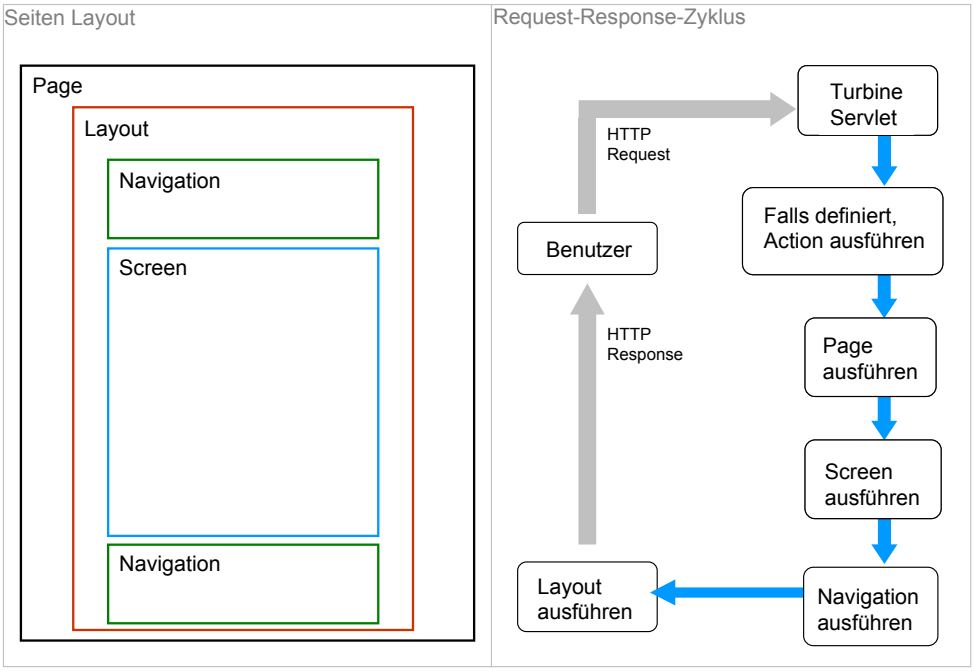


Abbildung 3: Seitenlayout und Request-Response-Zyklus von Turbine [Wang04]

Wenn für den Request eine Action definiert ist, wird sie von der Page gerufen. Ist dies der Fall wird von der Page das Layout Modul aufgerufen. Hier ist der Seitenaufbau festgelegt, der wiederum Navigation- und Screen- Module enthält. Die Navigation besteht aus einer Kopf- und Fußzeile, die auf jeder Seite angezeigt werden. Die eigentlichen Informationen werden im Screen dargestellt. Ein Screen stellt den Körper eines HTML-Dokumentes dar

und ist für die Benutzerinteraktion und Datenanzeige verantwortlich. Für die Präsentation werden Templating Frameworks, wie beispielsweise Velocity [Velocity04] vorgeschlagen. Es können aber auch JSPs zum Einsatz kommen.
Turbine besitzt eine komponentenbasierte Architektur. Module sind daher wieder verwendbar und leicht austauschbar. Für eine schnelle Einarbeitung in Turbine steht das Turbine Development Kit zur Verfügung. [Wang04]

Tabelle 2 zeigt einen Vergleich der drei Webframeworks in Bezug auf die Anforderungen, die das M2A-Projekt vorgibt.
Aufgrund der optimalen Unterstützung von XML und der leichten Verwirklichung von verschiedenen Ausgabeformaten wurde Cocoon als Webframework ausgewählt. Der modulare Aufbau von Cocoon erlaubt ein sehr flexibles Design von Webapplikationen und gewährleistet deren leichte Erweiterbarkeit. Einzelne Applikationen können auf dem Server sauber strukturierbar und vollkommen unabhängig von einander entwickelt werden.
Turbine sowie auch Struts haben ihre Schwächen in der einfachen Verwirklichung verschiedener Ausgabeformate. Auch bei der Unterstützung von XML bietet Cocoon mehr Möglichkeiten.

	Turbine	**Struts**	**Cocoon**
XML-Unterstützung	Turbine unterstützt die Verarbeitung von XML.	Bedingt, bei der Entwicklung stand die Verwendung von XML nicht im Vordergrund.	Cocoon erlaubt eine Optimale Verarbeitung von XML-Daten.
Ausgereift	ja	ja	ja
Verwirklichung Verschiedener Ausgabeformate	Möglich, aber ist nicht optimal unterstützt.	Ist zwar möglich, aber kein spezieller Mechanismus hierfür implementiert	Sehr gut, durch das Hinzufügen neuer Pipelines leicht möglich.
Adaption verschiedener Ausgabegeräte	Kein Mechanismus zur Erkennung von Clienten.	Kein Mechanismus zur Erkennung von Clienten.	Eine Möglichkeit zur Erkennung von anfragenden Clienten ist ansatzweise vorhanden.
Flexibel und leicht erweiterbar	Basieret auf Modulen, die wieder verwendbar sind. Als Grundgerüst dient, wie bei Cocoon, das Avalon Framework. Der Applikationsfluss wird in einer Konfigurationsdatei festgelegt.	Plugin- Mechanismus ab Version 1.1 ermöglicht es eigene Klassen zu erstellen. Der Applikationsfluss wird in einer zentralen Konfigurationsdatei festgelegt.	Basiert auf dem Avalon-Framework, Komponenten sind leicht austauschbar. Steuerung des Applikationsfluss über die Sitemap. Sehr komplexes, aber mächtiges Konzept

Tabelle 2: Vergleich von Turbine, Struts und Cocoon anhand der M2A-Anforderungen

Es ist auch teilweise möglich die verschiedenen Frameworks zu kombinieren. So kann man beispielsweise für die Formularbearbeitung Struts einsetzen und für das Generieren verschiedener Ausgabeformate auf Cocoon zurückgreifen. In dieser Arbeit sollte eine Lösung allerdings auf den Einsatz von einem Framework beschränkt werden.

3 Eingesetzte Technologien

Abbildung 4 zeigt einen abstrakten Überblick über das Design des M2A-Webservers auf Basis von Cocoon. Sie soll einige Anhaltspunkte geben, wo die Technologien, die in diesem Kapitel vorgestellt werden, zum Einsatz kommen.

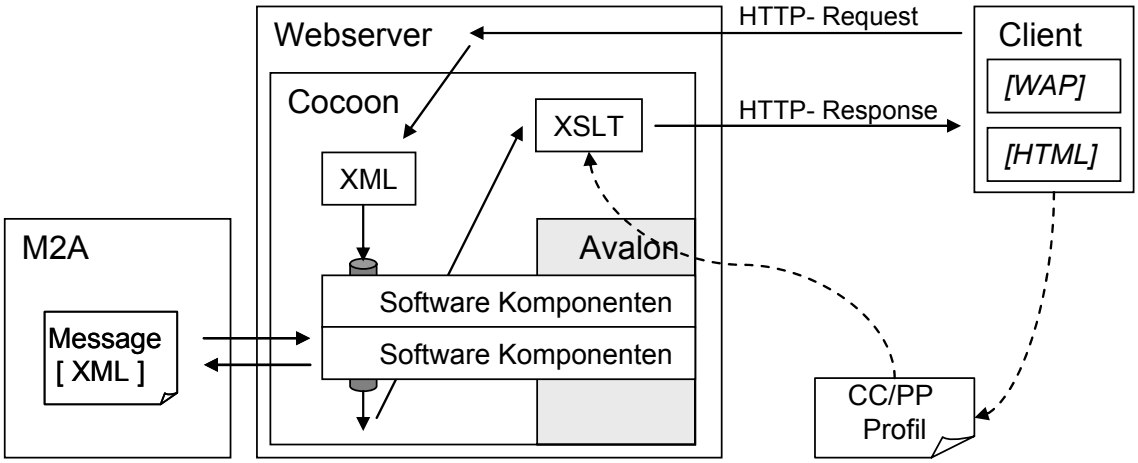


Abbildung 4: Übersicht eingesetzter Technologien

3.1 Extensible Markup Language und Familie

3.1.1 Extensible Markup Language

Mit der Extensible Markup Language kann die Syntax einer Auszeichnungssprache definiert werden. Unter einer Auszeichnungssprache versteht man Sprachen, die für die Beschreibung von Daten (ob Textverarbeitung, Webinhalt oder Grafik) konzipiert werden. Da XML nur für die Syntax verantwortlich ist und nicht für die Formatierung, wird diese separat in so genannten XSLT-Stylesheets definiert (mehr zu Stylesheets in Kapitel 3.1.3).
Ein XML-Dokument bildet eine Baumstruktur, welche mit einem XML-Parser (Kapitel 3.2.1 und 3.2.2) aufgebaut werden kann und somit der Zugriff einer Anwendung auf diese

Struktur ermöglicht wird. Abbildung 5 zeigt dieses Prinzip. XML-Dokumente bestehen aus Elementen (in der Baumstruktur als Knoten bezeichnet), die von den so genannten Tags begrenzt werden.

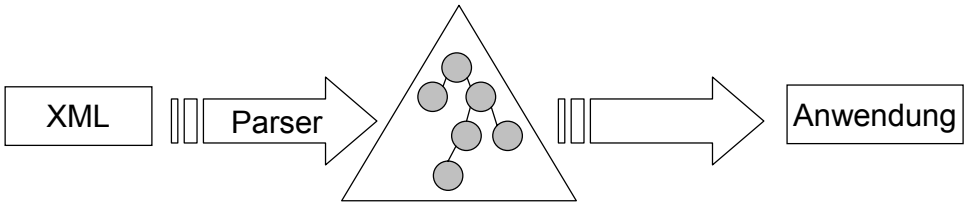
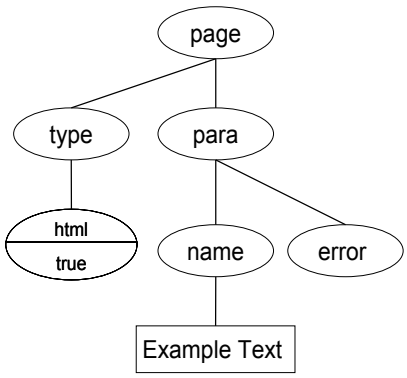


Abbildung 5: Aufbereitung der XML Daten über die Baumstruktur [RRZN01, Seite 10]

Beispiel 1 zeigt eine einfache XML-Struktur, in dem das Element `title` durch den Start-Tag `<title>` und den End- Tag `</title>` begrenzt ist. Das Element `<error>` ist ein leeres Element, es besitzt also keinen Inhalt, muss aber dennoch mit einem End- Tag abgeschlossen werden. Auch `<type>` in Zeile neun stellt ein leeres Element da, allerdings wurde hier eine Kurzschreibweise gewählt. Dieses Element enthält einen zweiten, wesentlichen Bestandteil von XML, das Attribut. Im Beispiel ist `html,` welches den Wert `true` besitzt, ein Attribut des Elements `<type>`. Attribute bieten die Möglichkeit Elemente genauer zu spezifizieren. Die erste Zeile des Beispiels zeigt den so genannten Prolog des XML-Dokumentes. Dieser teilt den Anwendungen, die dieses Dokument verarbeiten wollen mit, dass es sich um ein Dokument mit XML-Daten handelt.

```
1 <?xml version="1.0"?>
2 <page>
3 <title> Hello World </title>
4   <page>
5     <para>Example Text</para>
6     <error></error >
7    </page>
8    <type html="true"/>
9 </page>
```



Beispiel 1: XML-Dokument als Baumstruktur

Bei XML-Dokumenten wird zwischen einem wohlgeformten (well formed) und einem gültigem (valid) Dokument unterschieden.

Ein wohlgeformtes Dokument besteht, gemäß der XML-Spezifikation [XmlRec00], aus dem Prolog und mindestens einem Element. Jedes geöffnete Element muss auch wieder geschlossen werden. Ein Dokument, das nicht wohlgeformt ist, ist kein XML-Dokument.
Um ein gültiges Dokument zu erlangen, muss dieses zum einem wohlgeformt sein und hierüber hinaus auch formalen Anforderungen entsprechen, die an seine Struktur gestellt werden. Diese Anforderungen bzw. Regeln werden in Schemata festgelegt. Zwei Möglichkeiten zur Definition sind die Dokument Type Definition (DTD) oder die mächtigeren XML-Schemata. [RRZN01, Seite 9-13]

3.1.2 XML Namensräume

Namensräume (Namespaces) sind ein wichtiger Mechanismus in XML, um die Eindeutigkeit von Elementen zu erreichen. Jeder Namensraum muss deklariert werden bevor er verwendet wird. In Beispiel 2 geschieht dies in der dritten Zeile. Es wird hier ein Namensraum mit dem Präfix `max` deklariert. Diesem wird ein URI zugeordnet, die ihn weltweit eindeutig macht. Um den Namensraum auf ein Element anzuwenden, wird er mit einem Doppelpunkt von jenem getrennt. Die Anwendung des Namensraums demonstriert das Element `<size>`. Es kann zweifelsfrei entschieden werden, dass das Element `<max:size>` (Zeile fünf) für eine andere Größenangabe, wie `<size>` (Zeile sechs) zuständig ist. Die Elemente selbst sind identisch. Im Beispiel ist der Namensraum für alle Elemente, die innerhalb von `<object>` liegen gültig. Es besteht weiterhin die Möglichkeit einen Namensraum auf ein Element zu beschränken, mehrere Namensräume zu definieren oder durch eine Definition im Prolog des XML-Dokuments global verfügbar zu machen. [RZZN, Seite 72-75]

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!-- Definition des Namensraums max -->
 3 <object xmlns:max="http://www.example.org/myExample">
 4 <figure type="polygon">
 5   <max:size>33</max:size>
 6   <size>30</size>
 7 </figure>
 8 <figure type="elipse">
 9   <max:size>78</max:size>
10   <size>0</size>
11 </figure >
12 </object>
```

Beispiel 2: Definition und Verwendung von XML-Namensräumen

3.1.3 XSLT und XPath

Mit Hilfe der Extensible Stylesheet Language Transformation (XSLT) [XSLRec01] ist eine Umwandlung (Transformation) eines XML-Dokumentes möglich. Aus einem XML-Dokument kann mit Hilfe des entsprechenden XSLT-Dokumentes, dem so genannten Stylesheet, eine beliebige Darstellung der Daten generiert werden (z.B. HTML, PDF, WML, ...). Somit lässt sich die Trennung von Daten und deren Repräsentation verwirklichen. Die Transformierung selbst läuft in dem XSLT-Prozessor ab. Abbildung 6 soll dies verdeutlichen. Zwei der bekanntesten Prozessoren sind Saxon [Saxon04] und Xalan [ApacheXML04]. Xalan kommt in Cocoon (Kapitel 3.4) zum Einsatz.

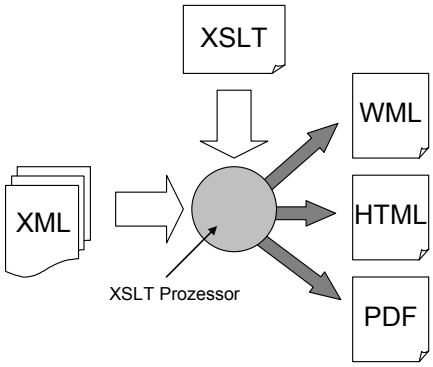


Abbildung 6: XSL Transformation

Ein XSLT-Stylesheet ist ein wohlgeformtes XML-Dokument (vergleiche Kapitel 3.1.1) und muss dessen Regeln genügen. Die Elemente in XSLT bezeichnet man als Templates (Vorlagen). Jede dieser Templates beschreibt ein Element des zu transformierenden XML-Dokumentes für das es gültig ist. In Beispiel 3 ist ein Ausschnitt aus einem XSLT-Stylesheet und dem zugehörigen XML-Dokument zu sehen. Die Templates werden mittels `<xsl:template>` definiert. Das optionale Attribut `match` definiert das Element oder Teildokument, auf das das Template angewendet werden soll. Mit dem Element `<xsl:value-of select="..."/>` ist ein Zugriff auf den Inhalt des Elementes möglich.

XSLT-Stylesheet

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <xsl:stylesheet version="1.0"
3                  xmlns:xsl="http://www.w3.org/TR/WD-xsl">
4
5  <-- Einstiegspunkt für die Verarbeitung -->
6  <xsl:template>
7    <xsl:apply-templates/>
8  </xsl:template>
9
10 <-- Template fuer das Element page -->
11 <xsl:template match="page">
12    <xsl:value-of select="/head"/>
13    <xsl:apply-templates select="size">
14 <xsl:template/>
15
16 <-- Template fuer das Element size -->
17 <xsl:template match="size">
18   <xsl:value-of select="@pt"/>
19 <xsl:template/>
```

XML-Dokument

```
<?xml version="1.0>
<page>
  <head>
    hello world
  </head>
  <size pt="10"/>
</page>
```

Beispiel 3: XSLT-Stylesheet für ein einfaches XML-Dokument

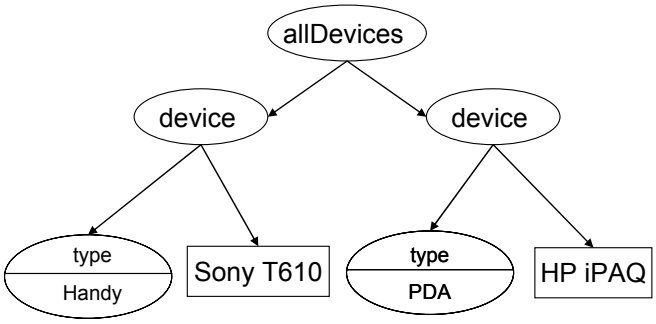
Die XML Path Language (XPath) [XPathRec99] stellt eine Möglichkeit dar, Teile eines XML Dokumentes zu adressieren bzw. anzusprechen. Es wird hierfür eine einfache, nicht XML-Syntax verwendet, die eine Navigation durch die hierarchische Struktur des Dokuments erlaubt. Zum Einsatz kommt XPath praktisch immer innerhalb eines XSLT-Stylesheets. Es kann beispielsweise geprüft werden, ob die von XPath adressierten Elemente auf die entsprechenden Templates zutreffen (Angabe eines XPath Ausdrucks in dem Attribut `match` in XSLT). [Geese01, Seite 113 – 140]

3.2 Parser

Ein Parser (auch Prozessor genannt) ist eine Software, die es ermöglicht Dokumente auszulesen, analysieren und einer weiteren Software zur Verfügung zu stellen.
Es gibt eine Vielzahl von Parsern die die folgenden zwei Konzepte Document Object Model (DOM) [DOM02] und Simple API (Application Programming Interface) for XML (SAX) [SAX04] implementieren. In dieser Arbeit wurde der SAX und DOM Parser des Apache Xerces2 [ApacheXML04] Projekts eingesetzt, da dies auch von Cocoon (Kapitel 3.4) verwendet wird.

3.2.1 Document Object Model

DOM ist ein Programmiersprachen unabhängiger Standard zur Bearbeitung von XML-Dokumenten. Anwendungen können auf XML-Dokumente zugreifen, verändern und erstellen. Die eingelesene XML-Datei wird intern als Baumstruktur im Speicher abgebildet. Dies ermöglicht zwar einen sehr komfortablen Zugriff auf die Elemente, führt aber bei größeren Dokumenten zu einer schlechten Performance.
Beispiel 4 zeigt die Abbildung eines XML-Dokumentes auf eine solche DOM-Baumstruktur.
In DOM ist es möglich Teilbäume zu extrahieren oder einzufügen. Im Vergleich zu SAX erlaubt es DOM, ganze Dokumente oder einzelne Elemente zu erzeugen und in die Baumstruktur einzufügen. [Seeboerger02, Seite 97-101]




```
1 <?xml version="1.0"?>
2  <allDevices>
3    <device type="Handy">
4      Sony T610
5    </device>
6    <device type="PDA">
7      HP iPAQ
8    </device>
9  </allDevices>
```

Beispiel 4 : Abbildung eines XML-Dokumentes in einem DOM Baum

3.2.2 Simple API for XML

SAX ist ein offener Standard, der Schnittstellen für XML-Parser definiert. Es gibt verschiedene XML-Parser die SAX Implementieren. SAX arbeitet ereignisgesteuert. Wenn der Parser innerhalb einer XML-Datei entsprechende Inhalte feststellt (z.B. Beginn eines Tags oder Ende des Dokuments), werden bestimmte Methoden, so genannte SAX-Events, aufgerufen. Die Summe aller Methodenaufrufe ergibt die Abbildung eines XML-Dokuments als SAX-Eventstream (SAX-Ereignisstrom). Beispiel 5 zeigt den SAX-Eventstream (rechts) eines XML-Dokumentes (links).

```
1 <?xml version="1.0"?>
2 <example>
3   <text>
4     Hello World
5   </text>
6 </example>
```

```
1 start document
2 start element: example
3 start element: text
4 characters: Hello World
5 end element: text
6 end element: example
7 end document
```

Beispiel 5: XML Darstellung und SAX- Eventstream

SAX durchläuft den XML-Baum sequenziell, und ist daher auch für große Dokumente geeignet, da im Vergleich zu DOM die Daten nicht im Speicher gehalten werden müssen. Der SAX-Parser spielt eine zentrale Rolle in Cocoon (Kapitel 3.4).
[Seeboerger02, Seite 27-35]

3.3 Composite Capabilities/Preference Profile

Das Composite Capabilities / Preference Profile (CC/PP) [CCPP04] ist ein vom World Wide Web Consortium (W3C) [W3C04] entwickeltes Framework zur standardisierten Beschreibung von Eigenschaften Mobiler Endgeräte.
Diese Eigenschaften, oder auch Fähigkeiten, werden in so genannten Profilen abgebildet. Profile werden im XML-Format dargestellt und bauen auf dem Resource Description Framework (RDF) [RDFRec04] auf. Durch das Aufkommen immer neuer Mobiler Endgeräte muss CC/PP flexibel und leicht erweiterbar sein.

3.3.1 Das Resource Description Framework

Die Beschreibung von Information aus dem World Wide Web war das Entwicklungsziel von RDF. Mit RDF lassen sich Metadaten, wie etwa Titel, Autor oder Erscheinungsdatum, von Ressourcen (z.B. Internetseiten) beschreiben. Eine grundlegende Eigenschaft von RDF ist es, dass seine Bestandteile über URIs identifiziert werden. Dies führt zu einem uneingeschränkt erweiterbaren Vokabular, was RDF zu einer idealen Grundlage für eine Beschreibungssprache Mobiler Endgeräte macht, da hier Flexibilität und Erweiterbarkeit sehr wichtig sind.

Informationen werden in RDF in so genannten Statements dargestellt. Ein solches Statement besteht aus einer Resource (Subject), welche Eigenschaften (Predicate) besitzt, die wiederum einen Wert (Object) haben. Solche RDF Statements lassen sich wie in Abbildung 7 als Graph darstellen.

Das Statement „*Arthur is the creator of the ressource http://www.example.org/nopanic"* teilt sich in folgendes Tripel auf:

- Subject: „*http://www.example.org/nopanic*". Dies ist die zu beschreibende Resource.
- Predicate: „*creator*". Die Eigenschaft, die die Beziehung beschreibt, in dem die Resource mit einem Object steht.
- Object: „ *Arthur* ". Der eigentliche Wert des Statements, auf die Subject und Predicate hinweisen. .

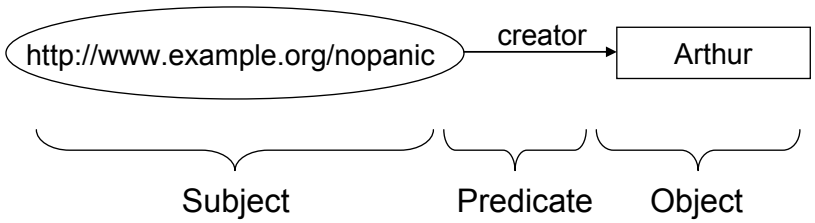


Abbildung 7: Ein einfaches RDF-Statement als Graph

Im Beispiel 6 ist die XML-Darstellung des Graphen zu sehen. Zur Unterscheidung der verschiedenen Vokabulare werden in Zeile drei und vier Namensräume deklariert. Die Bezeichnung `rdf` wird für den Namensraum des RDF-Vokabulars verwendet, der die Ressource darstellt (Zeile sechs). Analog hierzu wird der Namensraum für dieses Beispiel `ex` angegeben. Dieser beschreibt die Predicates (Zeile sieben). Das Object des Statements ist der Inhalt des Elementes `<ex:creator>`. [RDFRec04]

```
1 <?xml version="1.0"?>
2 <rdf:RDF
3     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4     xmlns:ex="http://description.org/exampleSchema/">
5
6     <rdf:Description about="http://www.example.org/nopanic">
7          <ex:creator>Arthur</ex:creator>
8      </rdf:Description>
9  </rdf:RDF>
```

Beispiel 6: Einfaches RDF-Beispiel als XML Darstellung

3.3.2 Aufbau eines CC/PP Profils

Im CC/PP Framework wird eine Vorlage für die Struktur eines Geräteprofils und die Anforderungen an ein konkretes Vokabular definiert. Grundsätzlich definiert CC/PP keinen Mechanismus zur Übertragung von Profilen, es wird auch kein konkretes Vokabular zur Beschreibung der Geräte festgelegt.
Um die Eigenschaften und Fähigkeiten eines Client einem Server bekannt zu machen, muss diesem das CC/PP Profil zugänglich gemacht werden. Bei der Verwendung des HTTP-Protokolls ist es nahe liegend, dass diese Informationen bei einem Request eines Clients mitgesendet werden. Der Server kann den Inhalt der Response mit Hilfe der CC/PP Profildaten anpassen. Diesen Mechanismus zeigt Abbildung 8.

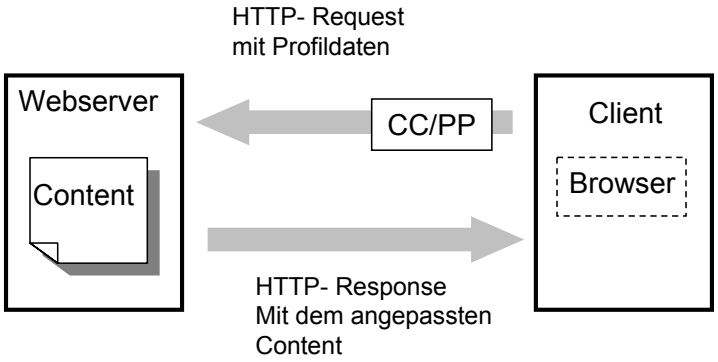


Abbildung 8: Übermittlung eines CC/PP Profil

Zurzeit unterstützt allerdings noch kein Browser die Übermittlung dieser Daten und zur Verarbeitung der Profile auf Serverseite gibt es momentan auch keine einsetzbare Lösung. Dies liegt zum einen sicherlich an der relativ neuen Technologie, zum anderen spielen hier aber auch Sicherheitsbedenken eine Rolle, da es sich um die Übermittlung personenbezogener Daten handelt.

Abbildung 9 zeigt einen schematischen Aufbau eines solchen Profils. Ein CC/PP Profil besteht aus einer Hierarchie von drei Ebenen. In der ersten Ebene werden die Hauptkomponenten (Components) beschrieben. In einer weitern Ebene befinden sich die Attribute (Attributes) zu diesen Components, wobei hier jede Component mindestens ein Attribute besitzt. Ein Beispiel für Component wäre die Hardware Plattform des Gerätes.

Hier sind zwei Attributes denkbar, Bildschirmhöhe und Breite, die das Display des Gerätes beschreiben. In der dritten Ebene sind die konkreten Werte (Values) zu sehen, die den Attributes zugeordnet sind.

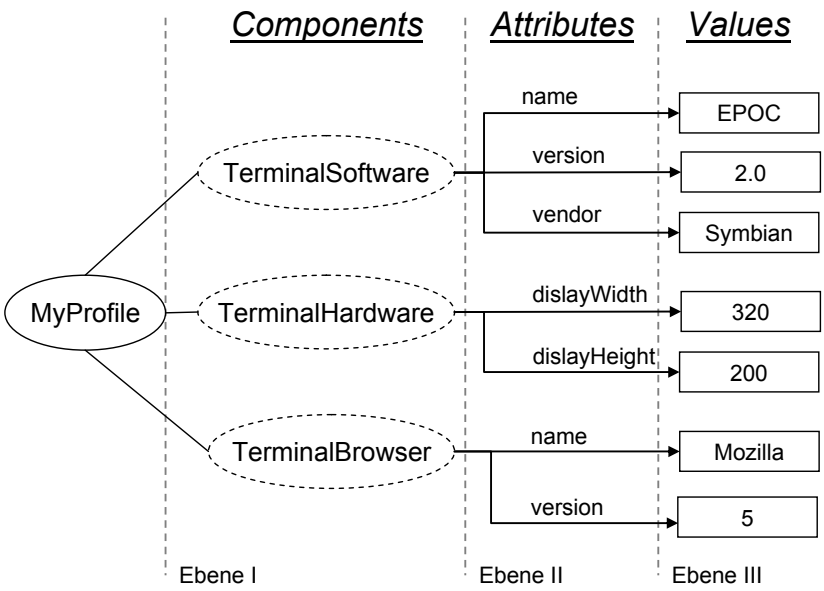


Abbildung 9: Schematischer Aufbau eines CC/PP Profils [CCPP04]

Beispiel 7 zeigt das CC/PP Profil aus Abbildung 9 in Form eines XML-Dokuments. Zu Beginn werden drei Namensräume durch URIs definiert (siehe Kapitel 3.1.2 XML Namensräume). In Zeile zwei wird die Benutzung von RDF deklariert. Der zweite Namensraum definiert ein CC/PP Kern Vokabular (Zeile drei), welches im Beispiel `component` enthält. Der dritte Namensraum beschreibt ein beispielhaftes konkretes Vokabular für die Attribute (Zeile vier). Die Werte der Attribute beinhalten die eigentlichen Eigenschaften des Endgerätes.

Wie schon erwähnt definiert CC/PP kein konkretes Vokabular, dies wurde hier zum Bessern Verständnis eingeführt. Ein Beispiel-Vokabular, das User Agent Profile, wird im folgenden Kapitel vorgestellt.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"
         xmlns:ex="http://www.ex.com/schema#">

  <rdf:Description
      rdf:about="http://www.ex.com/profile#MyProfile">

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.ex.com/profile#TerminalHardware">
        <rdf:type
        rdf:resource="http://www.ex.com/schema#HardwarePlatform"/>
        <ex:displayWidth>320</ex:displayWidth>
        <ex:displayHeight>200</ex:displayHeight>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.ex.com/profile#TerminalSoftware">
        <rdf:type
            rdf:resource="http://www.ex.com/schema#Software"/>
        <ex:name>EPOC</ex:name>
        <ex:version>2.0</ex:version>
        <ex:vendor>Symbian</ex:vendor>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.ex.com/profile#TerminalBrowser">
        <rdf:type
            rdf:resource="http://www.ex.com/schema#BrowserUA"/>
        <ex:name>Mozilla</ex:name>
        <ex:version>5.0</ex:version>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>
</rdf:RDF>
```

Beispiel 7: CC/PP Profil in XML [CCPP04]

Es besteht weiterhin die Möglichkeit in einem Profil Defaultwerte zu verwenden. In einem Szenario in dem fast identische Geräte bzw. Profile vorkommen, die sich beispielsweise nur in ihrer Speichergröße unterscheiden, kann man die identischen Daten in einem Defaultprofil speichern. Im eigentlichen Geräteprofil werden nur die Unterschiede hinterlegt und auf das Defaultprofil verwiesen. Dadurch wird die Bildung von redundanten Daten vermieden. [Jaboring04]

3.3.3 User Agent Profile

Wie in Kapitel 3.3.2 beschrieben, stellt das W3C mit dem CC/PP Profil kein Vokabular hierfür zur Verfügung. Das User Agent Profile (UAProf) [WAGUAProf01] schließt diese Lücke, indem es ein konkretes Vokabular einführt. In dieser Arbeit wurden Profilbeschreibungen mit diesem Vokabular verwendet, da hier schon über 250 Profile von 10 Herstellern [UAProfReposetory04] frei im Internet verfügbar waren.

Entwickelt wurde das UAProf von der Open Mobile Alliance (OMA) [OpenMobileAllinace02]. Dies ist ein Zusammenschluss aus ca. 300 Industriepartnern und hat sich zur Aufgabe gemacht, die Zusammenarbeit zwischen den Firmen zu fördern, Hürden zu beseitigen und die Einführung von Standards zu unterstützen.

Da jedes CC/PP Profil auch ein gültiges User Agent Profile ist, unterscheiden sich beide nicht in ihrem Aufbau. UAProf führt allerdings mehr unterschiedliche Components und Attributes ein. In Tabelle 3 werden alle Components und einige zugehörigen Attributes vorgestellt. Dies soll einen Eindruck über die vielfältigen Beschreibungsmöglichkeiten vermitteln. Insgesamt stehen 56 Attribute zur Beschreibung eines Mobilen Endgerätes zur Verfügung.

Components	**Beispiele für zugehörige Attributes**
HardwarePlatform	Informationen zur Bildschirmgröße, Farbtiefe, Hersteller oder auch Eingabemöglichkeiten, wie Tastaturumfang oder Touchscreen.
SoftwarePlatform	Akzeptierte Sound- und Videoformate, Information über den Hersteller des Betriebssystems und die Unterstützen Sprachen.
NetworkCharacteristics	Unterstützte Übertragungstechnologien (UMTS, GPRS,...), Verfügbarkeit oder die Latenz.
BrowserUA	Name des aktuellen Browsers, Möglichkeit des Ausführen von Javascripten auf dem Endgerät oder die Unterstützte HTML Version.
WapCharacteristics	Beschreibt beispielsweise die unterstützten WML Versionen.
PushCharacteristics	Bespiele hierfür sind die unterstützten MIME-Typen (Multipurpose Internet Mail Extension) oder die Größe der Push-Nachricht. Diese sind Daten die über WAP ohne eine direkte Anfrage auf das Endgerät gesendet werden. Beispielsweise die zurzeit beliebten Klingeltöne oder Logos.

Tabelle 3 : Komponenten und Beispielattribute im UAProf

3.4 Das XML Publishing Framework Cocoon

3.4.1 Grundlagen

Die Entwicklung von Cocoon begann im Jahre 1998 mit dem Ziel, die Webseiten der Apache Software Foundation von deren Inhalt und Layout zu trennen und besser zu strukturieren. Diese erste Version baute noch auf dem Document Object Model auf (siehe Kapitel 3.2.1). Diese Speicherintensive Verarbeitung führte zu einem Geschwindigkeitsverlust und somit zeichnete sich Cocoon in seiner ersten Version durch eine schlechte Performanz aus. Dies führte rasch zu einer Neuentwicklung, mit einem vollkommen überarbeiteten Design, die 2001 als Cocoon 2 veröffentlicht wurde.
Performance, Skalierbarkeit, Stabilität und Modularität waren die wichtigsten Punkte bei der Entwicklung von Cocoon 2, die auch in einer sehr guten Weise verwirklicht wurden. Die größte Neuerung bestand darin, die speicher- und zeitintensive Verarbeitung mit DOM durch SAX (vergleiche Kapitel 3.2.2) zu ersetzen.

Cocoon ist ein Java Framework mit einer offenen, komponentenbasierten Architektur. Grundlage für diese Architektur ist das Apache Projekt Avalon [Avalon04].
In diesem Framework wird die Erzeugung, Verwaltung und Anwendung von Komponenten definiert. Vorteile der Verwendung von Komponenten bestehen darin, dass somit eine wesentlich einfachere Wartung der Applikation erfolgen kann und sie einfach ausgetauscht werden können, ohne Änderungen in der Applikation selbst vornehmen zu müssen (Kapitel 3.5 beschäftigt sich ausführlicher hiermit).
Aus diesem Grund ist Cocoon auch einfach durch selbst entwickelte Komponenten zu erweitern. Auch die Verwendung von Komponenten aus anderen Projekten, die auf Avalon aufbauen, steht nichts im Wege. Ein Beispiel hierfür ist Apache Xerces [ApacheXML04] als XML-Parser oder Apache Xalan [ApacheXML04] als XSLT-Prozessor, die in Cocoon zum Einsatz kommen.
Die Arbeitsweise von Cocoon basiert auf dem klassischen Request-Response-Zyklus wie in Abbildung 10 zu sehen ist. Ein Request eines Clients wird von einem zentralen Servlet (Servlet Engine) entgegengenommen und direkt an Cocoon weitergereicht.

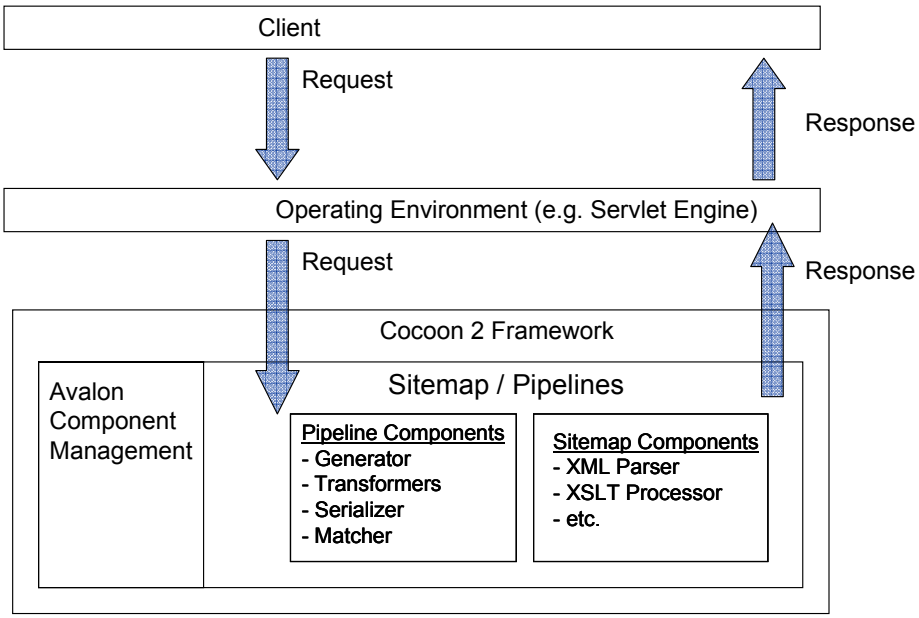


Abbildung 10: Die Arbeitsweise von Cocoon 2

Ein Antwortdokument wird von Cocoon generiert und an den Client zurückgeschickt. Jede Anfrage an Cocoon liefert eine Antwort in Form eines Dokuments. Ein Dokument besitzt eine eindeutige Bezeichnung (URI), die man in einem Browser angeben kann. Alle Informationen, die der URI „cocoon/“ folgen, beschreiben den Pfad des Dokuments innerhalb Cocoons.

Beispielsweise ist das Dokument helloWorld über http://localhost/cocoon/helloWorld erreichbar. Als Bindeglied zwischen Dokumentenname und Antwort fungiert die Sitemap, die im nächsten Kapitel vorgestellt wird. Der Präfix „cocoon/“ lässt sich selbstverständlich beliebig verändern bzw. komplett ausblenden. [Langham02]

3.4.2 Sitemap Komponenten und Pipelines

Die Sitemap stellt das zentrale Steuerelement von Cocoon da. Sie ist ein gültiges XML-Dokument (vergleiche Kapitel 3.1.1), welches die Arbeitsschritte für eine URL (Uniform Resource Locator) festlegt, die hier ausgeführt werden müssen.

Die grundlegende Struktur der Sitemap gliedert sich in zwei Teile. Zum ersten werden die Komponenten**,** welche für die Verarbeitung von Dokumenten nötig sind, definiert. Der zweite Teil listet die von Cocoon zu verarbeitenden Dokumente auf und verknüpft sie mit ihren URLs. Dies geschieht in so genannten Pipelines.

Die Pipeline kapselt alle Komponenten, die zur Verarbeitung eines oder mehrere Dokumente vom Request des Clients bis zur Response des Servers nötig sind.
Beispiel 8 zeigt eine fiktive Sitemap zur Verdeutlichung dieses Zusammenhangs. Es ist möglich für einzelne Projekte separate Sitemaps anzulegen. Diese lassen sich von einer zentralen Sitemap mounten, dadurch entsteht eine Strukturierung des gesamten Projektes.

```
1  <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
2     <map: components>                                          ┐
3        <map:generators/>                                       │
4        <map:transformers/>                                     │
5        <map:serializers/>                                      ├ Komponente
6        <map:matchers/>                                         │
7        ...                                                     │
8     </map:components>                                          ┘
9
10    <map:pipelines>
11      <map:pipeline>                                           ┐
12         <map:match pattern="helloWorld.xml">                  │
13            <!-- Die Verarbeitung der URL wird                 │
14                 hier von Komponenten durchgeführt -->         │
15         </map:match>                                          │
16         <map:match pattern="/myPath/helloSpace.xml ">         ├ Pipeline
17            <!-- Die Verarbeitung der URL wird                 │
18                 hier von Komponenten durchgefuehrt -->        │
19         </map:match>                                          │
20      </map:pipeline>                                          ┘
21    </map:pipelines>
22 </map:sitemap>
```

Beispiel 8: Aufbau der Sitemap

Die einzelnen Komponenten innerhalb der Pipeline tauschen ihre Daten über SAX-Eventstreams (vergleiche Kapitel 3.2.2) aus.
Abbildung 11 zeigt eine vereinfachte Darstellung eines Request-Response-Zyklus. Generator, Transformer und Serializer sind Komponenten innerhalb der Pipeline. Der Generator liest aus einer Quelle die Daten und generiert aus ihnen einen SAX-Eventstream. Transformatoren beeinflussen diesen und ein Serializer bringt den SAX-Eventstream in das entsprechende Ausgabeformat. Die Komponenten werden in den folgenden Kapiteln ausführlich vorgestellt.

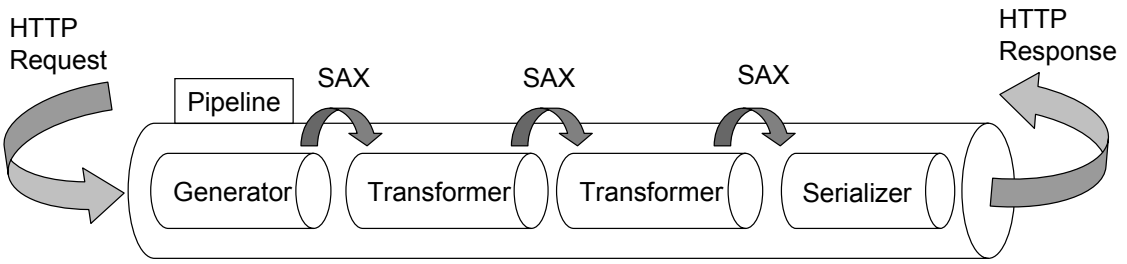


Abbildung 11: Request-Response-Zyklus einer Pipeline [Kumar04]

3.4.3 Komponenten innerhalb von Cocoon

Alle Komponenten werden in `<map:components>` definiert. Sie sind das „Handwerkszeug“ mit dem Cocoon seine Arbeit erledigen kann. Wie in Abbildung 12 zu sehen ist kann man eine grobe Unterteilung in Bearbeitungs- und Logikkomponenten vornehmen. Die erste Gruppe ermöglicht innerhalb der Sitemap eine Bearbeitung bzw. Manipulation des SAX-Eventstreams. Die zweite Gruppe ist für die Ablaufsteuerung und der Auflösung der URIs zuständig. Eine Action Komponente ist ein Sonderfall. Sie ist an keine konkrete Aufgabe gebunden. Da die Architektur von Cocoon Komponentebasiert ist, ist es leicht möglich eigene Komponenten zu entwickelt, und somit auf spezielle Anforderungen zu reagieren. Im Zuge dieser Arbeit wurden solche Komponenten erstellt. Im Kapitel 3.5 wird beschrieben, wie man bei einer Eigenentwicklung vorgehen muss.

Da Cocoon eine Vielzahl verschiedener Implementierungen dieser Komponenten bereits enthält, werden in den folgenden Abschnitten hierfür nur Beispiele genannt. Für einen vollständigen und aktuellen Überblick aller Implementierungen wird an dieser Stelle auf die Benutzerdokumentation [CocoonDoc04] verweisen.

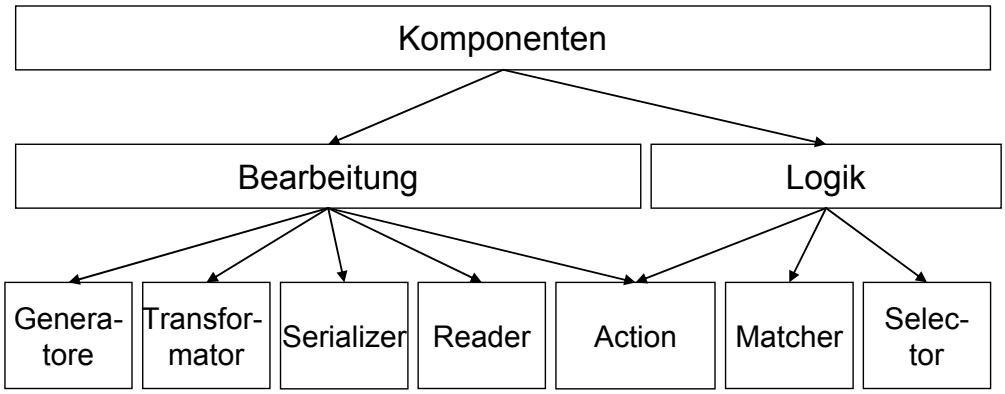


Abbildung 12: Übersicht der Komponentengruppen in Cocoon [Niedermeier03]

Generator

Am Anfang jeder Pipeline steht ein Generator, der für die Erzeugung des SAX-Eventstreams zuständig ist. Dieser wird anderen Komponenten, wie Transformern oder Serializern zur Verfügung gestellt. Cocoon beinhaltet schon eine Reihe von Generatoren (ca. 16), die für verschiedene Anwendungsfälle zugeschnitten sind.

Transformer

Transformer sind für die Manipulation des SAX-Eventstreams und die Weitergabe an die folgende Komponente zuständig. Es können beliebig viele Transformer in einer Pipeline vorkommen. Auch hier ist Cocoon schon mit einer großen Anzahl von Transformatoren ausgestattet.

Serializer

Am Ende einer Pipeline muss ein Serializer stehen, der den SAX-Eventstream in das gewünschte Ausgabeformat umwandelt. Des weitern wird der MIME-Type für die Response festgelegt. Cocoon liefert für praktisch jede Anwendung einen passenden Serializer.

Reader

Ein Reader ist ein Generator und Serializer in einem. Eine angegebene Resource wird direkt eingelesen und ohne eine Transformation zum Client weitergeleitet. Eine

Umwandlung in einen SAX-Eventstream wird umgangen. Dies ist nötig, wenn Dateien (z.B. Grafiken) unverändert zum Client gesendet werden sollen.

Selector

Mit Hilfe des Selectors können Verzweigungen innerhalb einer Pipeline anhand bestimmter Bedingungen realisiert werden. Hiermit ist es möglich, den SAX-Eventstream in verschiedene Verarbeitungswege zu leiten. Man kann seine Arbeitsweise mit der „if – then – else“ Anweisung aus Programmiersprachen vergleichen.

Matcher

Matcher stellen einen Hauptbestandteil von Pipelines dar. Sie ermöglichen es einen bestimmten Teil einer Pipeline auszuwählen der verarbeitet werden soll. Am häufigsten werden sie in Verbindung mit der anfragenden URI eingesetzt, um dies auf deren zugehörigen Dokumente abzubilden.

Beispiel 9 zeigt die Verwendung eines solchen Matchers. Dieser Matcher erlaubt auch die Verwendung von regulären Ausdrücken. Somit ist es möglich eine Struktur innerhalb der Pipelines aufzubauen, die sehr flexibel auf eingehende URIs reagieren kann und in die neue Dokumente ohne Anpassungen integriert werden können.

```
1  ...
2    <map:match pattern="login" type="wildcard">
3      <!-- Verarbeitung der URI http://localhost/cocoon/login -->
4     </map:act>
5       <!-- Verarbeitung aller Requests die mit der URI
6            http://localhost/cocoon/data/ beginnen.-->
7       <map:match pattern="data/*" type="wildcard">
8
9       <!-- Über den Ausdruck {1} ist der Zugriff auf den Inhalt des
10           Wildcards möglich. -->
11      <map:generator type="file" src={1}/>
12       ...
13    </map:act>
14 ...
```

Beispiel 9: Anwendung eines Matchers mit Regulären Ausdrücken

Action

Actions sind Komponenten, die an keine speziellen Aufgaben gebunden sind. In ihnen kann man eine sehr aufwendige Logik implementieren, allerdings besitzen sie keinen Zugriff auf den SAX-Eventstream der Pipeline. Die Action stellt zusammen mit dem

Transformator die mächtigsten und flexibelsten Komponenten von Cocoon dar. Für den praktischen Teil dieser Arbeit wurden mehrere eigene Action- und Transformator Komponenten entwickelt. In Kapitel 3.5.3 wird die Vorgehensweise bei der Entwicklung neuer Komponenten beschrieben.

Die Pipeline

Die Pipeline ist der Ort an dem alle Komponenten zum Einsatz kommen. Im einfachsten Fall besteht eine Pipeline aus einem Generator, beliebig vielen Transformatoren und einem Serializer. Dieser Fall wird im Beispiel 10 betrachtet.

```
 1 <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
 2
 3 <!-- Definition der Komponenten -->
 4 <map:components>
 5    <map:generators default="file">
 6     <map:generator name="file" src="...generation.FileGenerator"/>
 7    </map:generators>
 8
 9    <map:transformers default="xslt">
10     <map:transformer  name="xslt"
11                       src="...transformation.TraxTransformer">
12    </map:transformers>
13
14    <map:serializers default="html">
15     <map:serializer mime-type="text/html" name="html"
16                     src="...serialization.HTMLSerializer">
17    </map:serializers
18 </map:components>
19
20 <!-- Verwendung der Komponenten -->
21 <map:pipelines>
22    <map:pipeline>
23      <map:match pattern="hello">
24        <map:generate type="file" src="hello/hello.xml"/>
25        <map:transform src="hello/hello.xslt"/>
26        <map:serialize type="html"/>
27       </map:match>
28     </map:pipeline>
29 </map:pipelines>
30
31 </map:sitemap>
```

Beispiel 10: Auszug aus einer Sitemap

In Zeile 4-18 der Sitemap werden innerhalb des Elements `<map:components>` alle Komponenten festgelegt, die für die Verarbeitung des Dokuments nötig sind. In Zeile fünf

wird ein Generator definiert. Das Attribut `src` legt den Pfad fest, in dem Cocoon den Generator finden kann, `name` spezifiziert eine eindeutige Bezeichnung, mit der man innerhalb der Sitemap diesen Generator ansprechen kann. Analog hierzu wird in Zeile neun ein Transformer bzw. in Zeile 14 ein Serializer definiert. Grundsätzlich können beliebig viel, d.h. unterschiedliche Komponenten, festgelegt werden. Durch das Attribut `default` lässt sich bei mehreren Komponenten diejenige festlegen, die standardmäßig verwendet wird.
Im zweiten Teil des Beispiels erfolgt die eigentliche Verarbeitung des Requestes. Ein Request, die auf das Muster (Pattern) in Zeile 23 zutrifft, wird von den Komponenten innerhalb des `<map:match pattern="...">` Elements verarbeitet.
Die Datei `hello.xml` wird eingelesen und ein SAX-Eventstream gebildet (Generator). Dieser wird mit einem XSLT-Stylesheet in ein HTML-Format gebracht (Transformer). Zum Schluss wird der transformierte Inhalt des Dokumentes nach HTML serialisiert (Serializer). Das Ergebnis wird nun von Cocoon als Response an den Client zurückgeliefert. [Niedermeier03]

Extensible Server Pages

Die Extensible Server Pages (XSP) fand in dieser Arbeit zwar keine Anwendung, dennoch möchte soll hier kurz dieses interessante Konzept vorstellen und aufzeigen, warum XSPs nicht eingesetzt wurden.
Vereinfacht gesagt ist XSP ein XML-Dokument, in dem Logik eingebettet ist. XSP wurde speziell für Cocoon entwickelt. Das Konzept ist vergleichbar mit anderen Skriptsprachen wie JSP, Active Server Pages (ASP) oder auch Hypertext Preprocessor (PHP). Ein klarer Unterschied besteht darin, dass XSP nicht nur in HTML eingebettet werden kann, sondern auch in jedem beliebigen XML-Dokument.
Jede XSP-Datei wird durch einen speziellen Generator interpretiert. Dieser Generator wandelt die XSP-Datei in eine Java Klasse um. XSPs können zur Laufzeit verändert werden. Innerhalb der Sitemap wird das XSP-Dokument mit Hilfe des entsprechenden Generators identisch zu einem XML-Dokument behandelt (Vergleiche Beispiel 10, Zeile fünf).

Beispiel 11 zeigt ein XSP das zwei Zahlen addiert. Man kann hier sehen, dass XSP ein striktes Separation of Concerns nicht einhält. Der Content und die Logik werden hier

vermischt. Aus diesem Grund wurde in dieser Arbeit auf die Verwendung von XSP verzichtet. Die komplette Logik kann in Cocoon auch in selbst entwickelten Komponenten verwirklicht werden. Der Arbeitsaufwand hierfür ist zwar erheblich größer, allerdings kann ein sauberes SoC eingehalten werden. Des Weiteren können die Vorteile einer Komponentenbasierten Softwareentwicklung (siehe Kapitel 3.5) ausgeschöpft werden.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <xsp:page language="java" xmlns:xsp="http://apache.org/xsp">
4
5    <page>
6      <title>Hello World</title>
7      <content>
8        <para>Das Ergebnis sollte 42 sein!</para>
9
10        <!-- Hier beginnt der XSP Teil -->
11        <xsp:logic> int i = 40 + 2</xsp:logic>
12        <xsp:expr> i </xsp:expr>
13     </content>
14   </page>
15 </xsp:page>
```

Beispiel 11: Ein einfaches Extensible Server Page (XSP) Beispiel

Den Umstand der Vermischung von Content und Logik haben auch die Entwickler von Cocoon erkannt und bieten für XSP eine Lösung, um dies zu umgehen, die so genannten Logiksheets. Dies sind XSLT-Stylesheets, die auf das XSP-Dokument angewendet werden können. Die Logiksheets enthalten die gesamte Logik aus dem eigentlichen XSP-Dokument. Das Konzept kann allerdings bei größeren Projekten unübersichtlich werden und das Debugging des Source-Codes ist nicht so komfortabel möglich, wie man es bei reinen Java-Klassen in einer modernen Entwicklungsumgebung zu schätzen gelernt hat.
[Brogden03, Seite 169-171]

3.5 Erstellen neuer Komponenten für Cocoon

Wie schon bei der Vorstellung von Cocoon in Kapitel 3.4 beschrieben, basiert es auf dem Avalon Projekt. In diesem Kapitel wurden schon viele Komponenten vorgestellt, die innerhalb von Cocoon zum Einsatz kommen, und eine Vielzahl von Anwendungen abdecken. Dennoch ist es oft nötig eigene Komponenten zu erstellen bzw. bestehende zu erweitern. Hier soll nun gezeigt werden, wie solche Komponenten entwickelt werden und eingesetzt werden. Nach diesem Muster wurden die Komponenten im praktischen Teil dieser Arbeit verwirklicht (Kapitel 5).

3.5.1 Das Avalon Framework

Avalon ist ein Open Source Projekt der Apache Software Foundation und stellt ein Java Framework zur einheitlichen Entwicklung und Nutzung von Softwarekomponenten zur Verfügung.

Komponenten können helfen, einige Probleme in der Softwareentwicklung zu lösen oder zumindest zu vereinfachen. Durch ihre Verwendung wird die Struktur einer Anwendung leichter verständlich und Veränderungen einzelner Teile verursachen keine ungewünschten Seiteneffekte. Auch die Entwicklungszeit neuer Anwendungen ist geringer, da man auf bestehende Komponenten aus anderen Projekten zurückgreifen und sie unverändert wieder einsetzen kann. Avalon verwendet zwei zentrale Konzepte, um diese Eigenschaften zu realisieren. Die Entwurfsmuster (Design Patterns) Inversion of Control (IoC) und Separation of Concerns (SoC) werden hierfür eingesetzt.

Bei Inversion of Control (Umkehrung der Kontrolle) wird die Kontrolle von einer zentralen Stelle übernommen. Mit anderen Worten: Das Framework übernimmt die Erzeugung, Konfiguration, Verwaltung und Zerstörung der Komponenten. Dies steht im Gegensatz zur klassischen Programmierung, in der die Anwendung dies kontrolliert.

Separation of Concerns (Trennung der Zuständigkeiten) besagt in Avalon, dass genau definiert werden muss, welche Aufgaben jede Komponente erfüllen kann. Realisiert wird dies in Avalon durch so genannte Maker Interfaces. Soll eine Komponente in der Lage sein sich mit einem bestimmten Bereich zu befassen, so implementiert man einfach das entsprechende Interface. Ein Beispiel ist das Interface `Configurable`. Eine Komponente, die dieses Interface implementiert, ist konfigurierbar und erhält bei ihrer Instantiierung

vom Framework entsprechende Konfigurationsinformationen. Benötigt man die Möglichkeit der Konfiguration nicht, wird das Interface einfach nicht implementiert.

Avalon teilt sich in verschieden Unterprojekte auf, wobei Cocoon nur die Teile LogKit, Excalibur und Framework benutzt.
Framework bildet das Fundament. Es definiert die Schnittstellen und Konventionen, die für eine Komponentenbasierte Softwareentwicklung nötig sind und stellt einige Standard-Implementierungen zur Verfügung.
Das Excalibur Projekt definiert eine Reihe von Komponenten, die häufig benötigt werden. Eigene Anwendungen können auf diese zurückgreifen. Beispiele hierfür sind die Unterstützung bei der Verarbeitung von XML oder das Auflösen von URIs (Source Resolving) auf das Dateisystem des Servers. Cocoon benutzt sehr viele dieser Funktionalitäten in seinen Komponenten.
Bei dem Unterprojekt LogKit handelt es sich um eine schnelle und sehr mächtige Logging-API, welche im gesamten Avalon Projekt zum Einsatz kommt. [Ziegler03]

3.5.2 Entwicklung eigener Komponenten für Cocoon

Das Vorgehen bei der Entwicklung einer Komponente wird nun in diesem Kapitel anhand eines Beispiels gezeigt. Folgende drei Schritte sind hierbei nötig:

1. Definition eines Interfaces das den Typ und Verwendungszweck der Komponente beschreibt. In Avalon spricht man von einer so genannten Rolle (Role).
2. Es muss mindestens eine Implementierung dieses Interface geben.
3. Registrierung der Komponente bzw. der Role in einer zentralen Konfigurationsdatei.

Die Quellcode-Beispiele 12 bis 15 zeigen diese Schritte. In Beispiel 12 wird ein Interface definiert, das die Funktionalität der Komponente beschreibt und die Rolle festlegt (hier `examples.avalonExample.HelloWorld`). Es ist üblich, hier den kompletten Klassennamen des Interfaces zu verwenden. Diese Komponente besitzt nur die Methode `hello`, die ein „Hello World!“ auf der Konsole ausgibt.

```
1 package examples.avalonExample;
2
3 import org.apache.avalon.framework.component.Component;
4
5  public interface HelloWorld extends Component {
6     String ROLE ="examples.avalonExample.HelloWorld";
7     public void hello();
8  }
```

Beispiel 12: Festlegung eines Interfaces mit seiner Rolle

Der Quellcode in Beispiel 13 zeigt die konkrete Implementierung dieses Interfaces. Die einzige Methode, die das Interface aus dem vorherigen Beispiel angibt, wird nun ausimplementiert.

```
1 package examples.avalonExample;
2
3 public class HelloWorldImpl implements HelloWorld {
4
5  public void hello() {
6     System.out.println("Hello World!");
7  }
8 }
```

Beispiel 13: Implementierung des Interfaces aus Beispiel 12

Nun fehlt noch die Registrierung der Komponente, um sie in Cocoon einsetzen zu können. Cocoon besitzt hierfür eine zentrale Konfigurationsdatei, in die alle Komponenten eingetragen werden.
Beispiel 14 zeigt einen Ausschnitt hieraus. Die Komponente wird mit ihrer konkreten Implementierung und ihrer Rolle eingetragen, und ist nun bereit, innerhalb von Cocoon verwendet zu werden. Es ist weiterhin möglich ihr einen Logger zuzuweisen. Die Verwendung des Loggers in einer Komponente wird im Kapitel 5.3.5 beschrieben.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <cocoon version="2.1">
3   ...
4 <component class ="examples.avalonExample.HelloWorldImpl"
5            role  ="examples.avalonExample.HelloWorld"
6            logger="exampleLogger"/>
7   ...
8 </cocoon>
```

Beispiel 14: Registrieren einer Komponente in der Cocoon Konfigurationsdatei

Da bei Avalon (und dadurch auch bei Cocoon) das Konzept der Inversion of Control umgesetzt wurde, darf eine Komponente niemals direkt instantiiert werden. Dies muss immer über das Framework geschehen, da es sonst nicht die Kontrolle über die Komponente besitzt. Diese Aufgabe übernimmt der Service Manager. Er verwaltet alle verfügbaren Komponenten und nimmt eine zentrale Rolle innerhalb des Frameworks ein.
Beispiel 15 zeigt diesen Mechanismus.
Man erhält einen Service Manager durch Implementieren des Interfaces `Serviceable`. Durch den Aufruf der Methode `lookup` (Zeile 13), bei der als Übergabeparameter ein Rollennamen angegeben wird, erhält man die Komponente, die mit dieser Rolle assoziiert wird. Nur der Service Manager kennt die Implementierung, die hinter der Rolle bzw. dem Interface steht. Die Anwendung, die diese Komponente verwendet, ist hiermit unabhängig von deren tatsächlichen Implementierung. Somit kann eine andere Implementierung der Komponente verwirklicht werden, ohne die Anwendung umzuschreiben. Es wird erst zur Laufzeit entschieden, welche Implementierung verwendet wird. Man nennt diesen Mechanismus „Late Binding".
Da der Service Manager nicht nur für die Erzeugung der Komponenten verantwortlich ist, sondern auch für deren Zerstörung, ist es sehr wichtig ihm die Komponenten über die Methode `release` wieder zurückzugeben. Dies wurde in Zeile 20 in einer `finally` Klausel eingebettet. Somit kann man garantieren, dass die Komponente auch im Falle eines Fehlers an den Service Manager zurückgegeben und sauber aufgeräumt wird.

```
1  import examples.avalonExample.HelloWorld;
2  import org.apache.avalon.framework.service.ServiceManager;
3  import org.apache.avalon.framework.service.ServiceException;
4
5  public class AvalonTest{
6
7  public void testExample() {
8
9        HelloWorld myHelloWorld = null;
10
11       try {
12         //Instanziieren der Komponente über den Servicemanager
13         myHelloWorld = this.manager.lookup(HelloWorld.ROLE);
14         System.out.println( myHelloWorld.hello() );
15       } catch(ServiceException e) {
16          // Exception handling
17       }
18
19       // Rueckgabe der Komponente an den ServiceManager
20       finally {
21          this.manager.release(myHelloWorld);
22               }
23       }
24    }
```

Beispiel 15: Verwendung des ServiceManagers

Die Komponente, die im Beispiel entwickelt wurde, stellt nur eine Basisfunktionalität zur Verfügung; sie gibt einen Text aus. Um nun auf Funktionalitäten zurückgreifen zu können, die das Avalon Framework bereitstellt, muss sie, wie schon erwähnt, entsprechende Maker Interfaces implementieren.

Die Phase des Erstellens und Aufräumen von Komponenten bezeichnet man als Lebenszyklus (Lifecycle). Innerhalb des Lebenszyklus überprüft das Framework, welche Marker Interfaces eine Komponente implementiert. Daran geknüpft ruft es entsprechende Methoden auf, die wiederum eine Aktion auslösen. Die Beispielkomponente wird nun dahingehend erweitert, dass sie einen Text ausgibt, der bei ihrer Instantiierung angegeben werden kann.

In Beispiel 16 ist die Komponente zu sehen die zusätzlich das Interface `Parameterizable` implementiert. Sie kann nun Parameter verwenden, die beim Eintragen der Komponente in die zentrale Konfigurationsdatei Cocoons, wie in Beispiel 17, angegeben werden können. Wird diese Komponente erstellt ruft der ServiceManager, als Aktion auf das Marker Interface, die Methode `parameterize` und übergibt ihr alle Parameter die zuvor entsprechend angeben wurden. [Ziegler03]

```
1  package examples.avalonExample;
2
3  public class HelloWorldImpl implements HelloWorld, Parameterizable
4  {
5
6  private String text;
7
8   public void hello() {
9      System.out.println( text );
10  }
11
12  // Methode des Maker Interfaces, wird vom ServiceManager gerufen
13  public void Parameterizable(Parameters para)
14              throws ParameterException{
15   text = para.getParameter("myText");
16  }
17 }
```

Beispiel 16: Benutzung eines Maker Interfaces in einer Komponente

```
1  <component class ="examples.avalonExample.HelloWorldImpl"
2            role  ="examples.avalonExample.HelloWorld"
3            logger="exampleLogger">
4  <!—Angabe der Parameter, die die Komponente verarbeiten kann -->
5        <parameter name="myText" value="Hallo Welt!"/>
6  </component>
7  </cocoon>
8
```

Beispiel 17: Angabe des Parameters in der Cocoon Konfigurationsdatei

3.5.3 Sitemap Komponenten erstellen

Da Komponenten innerhalb Cocoons auf dem Avalon Framework basieren, gelten für sie auch alle Vorgaben, die im Kapitel zuvor beschrieben wurden. Um das Erstellen von Komponenten, die innerhalb der Sitemap zum Einsatz kommen, zu erleichtern, bietet die sehr umfangreiche API von Cocoon [CocoonAPI04] schon eine Vielzahl abstrakter Basisklassen. Diese kapseln einen großen Teil der Komplexität, die das Komponentenmodell von Avalon mit sich bringt. Ein Entwickler kann durch Generalisierung von diesen Klassen schnell eigene Komponente verwirklichen.

Die Entwicklung einer Sitemap Komponente soll anhand einer Action, deren Aufgaben innerhalb Cocoons in Kapitel 3.4.3 vorgestellt wurde, demonstrieren. Sie ist die wohl am häufigsten selbst entwickelte Sitemapkomponente. Im praktischen Teil dieser Arbeit kamen vier selbst entwickelte Actions für das Verarbeiten von Profildaten und die Benutzerauthentifikation zum Einsatz (Kapitel 5.3.1 und 5.3.2). Hierüber hinaus wurden mehrere Transformer Komponenten entwickelt (Kapitel 5.3.4). Das Grundprinzip bei der Entwicklung bleibt allerdings bei allen Sitemap Komponenten gleich.
Das Beispiel 18 zeigt eine einfache Action. Auf die „import“ Anweisungen wurde aus Gründen der Übersichtlichkeit verzichtet.

```
1  public class LoginAction extends AbstractAction {
2
3        public Map act(Redirector redirector,
4                       SourceResolver sourceResolver,
5                       Map objectModel, String src,
6                       Parameters parameters) throws Exception {
7
8              //Das Request Objekt aus dem objectModel
9              request = ObjectModelHelper.getRequest(objectModel);
10
11             //Die Wert der Parameter username und passWord auslesen
12             String userName = request.getParameter("userName");
13             String passWord = request.getParameter("passWord");
14
15             boolean success = registerUser(userName, passWord);
16
17             if(success == true){
18                 Session session = request.getSession();
19                 session.setAttribute(“username”,username);
20                 return objectModel;
21             } else {
22                 return null;
23             }
24       }
25 }
```

Beispiel 18: Erweiterung einer Action

Durch das Generalisieren von der Cocoon Basisklasse `AbstractAction` wird die Methode `act` vererbt (Zeile 3). Diese wird von Cocoon aufgerufen wenn die Action innerhalb einer Pipeline ausgeführt wird. Diese Methode wird überschrieben, und erlaubt so das Ausführen von eigenen Anweisungen. Interessant sind die Parameter von `act`, auf die man zurückgreifen kann.

- `Redirector redirector`: Dieses Object erlaubt die Weiterleitung eines Requests zu einer beliebigen URI.

- `SourceResolver sourceResolver`: Dokumente können von beliebigen Ressourcen geladen werden (Eine Erweiterung der `SourceResolver` Komponente aus dem Excalibur Projekt).
- `Map objectModel`: Diese Map repräsentiert verschiedene Umgebungsobjekte wie beispielsweise den aktuellen Request. Den Zugriff auf diese Informationen erhält man über die Klasse `ObjectModelHelper`.
- `String src`: Zugriff auf das optionale Attribut `src`, das man bei der Verwendung dieser Komponente in einer Pipeline angeben kann.
- `Parameters parameters`: Variablen, die der Action innerhalb der Sitemap übergeben werden können.

Die Implementierung des Beispiels ist einfach gehalten. Zunächst wird mit Hilfe des `ObjectModelHelper`, den jede Sitemapkomponente besitzt, das aktuelle Requestobjekt aus dem `objectModel` ausgelesen. Die Werte der Request Parameter `userName` und `passWord` werden in den Zeilen 12 und 13 abgefragt. Sie werden der fiktiven Methode `registerUser` übergeben. Wenn die Überprüfung nicht erfolgreich war, wird `null` zurückgegeben (Zeile 22), die weitere Verarbeitung der Action wird abgebrochen. Ist die Überprüfung jedoch erfolgreich, wird das `objectModel` (Zeile 20) als Rückgabewert verwendet. Zuvor wird mit `getSession` das zugehörige Session Objekt des Requestes geholt. Der Name des Benutzers wird als ein Attribut dieser Session gesetzt.
Jede Sitemapkomponente hat über den `ObjectModelHelper` Zugriff auf das aktuelle Request Objekt, und hierüber auf das Session Objekt dieses Requests. Somit stehen die Daten, die in der Session gespeichert werden, allen Komponenten in der Sitemap zur Verfügung.

Die Action ist nun bereit zum Einsatz innerhalb einer Pipeline. Zunächst muss sie in der Sitemap registriert werden wie in Beispiel 19 zu sehen. Diese geschieht im `<components>` Abschnitt der Sitemap. Sie kann danach beliebig oft in verschiedenen Pipelines eingesetzt werden. Im Beispiel verzweigt der Request in Abhängigkeit vom Ergebnis der Action mit der Anweisung `<redirect-to uri="...">` , in den Zeilen 9 und 11, zu unterschiedlichen URIs. Ist die Registrierung des Benutzers erfolgreich (Rückgabewert der Action ist ungleich `null`), gelangt er zur URI `protected`. Schlägt die Anmeldung fehl (Rückgabewert der

Action gleich `null`) wird er zu der URI `errorPage` umgeleitet. Hinter diesen URIs kann wieder eine weitere, beliebig komplexe, Logik stehen. Diese kann den Request weiter verzweigen lassen oder auch direkt die Daten für eine Response generiert.

```
1  <!-- Component Abschnitt der Sitemap -->
2  <map:actions>
3   <map:action name="loginAction" src="example.cocoon.LoginAction"/>
4  </map:actions>
5
6  <!-- Pipeline Abschintt der Sitemap -->
7  <map:match pattern="login">
8    <map:act type="loginAction">
9      <redirect-to uri="protected"/>
10   </map:act>
11     <redirect-to uri="errorPage"/>
12 </map:match>
```

Beispiel 19: Registrierung und Verwendung einer eigenen Action innerhalb der Sitemap

Eine Registrierung der Action in der Konfigurationsdatei von Cocoon, wie in Beispiel 14 zu sehen, ist nicht nötig, da keine Komponente komplett neu entwickelt wurde, sondern eine bestehende erweitert. Alle Actions von Cocoon implementieren das Interface `Action`. Hier ist die Rolle für den Avalon Service Manager festgelegt und die Methode `act` definiert. [Niedermeier04, Seite 393-439]

4 Das Message to Anywhere Projekt

Message to Anywhere Framework [M2A04] wird aktuell am Fraunhofer Institut für Produktionstechnik und Automatisierung (IPA) in Zusammenarbeit mit dem Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart (IAS) und fünf Projektpartnern aus der Industrie entwickelt. Gefördert wird das Projekt vom Bundesministerium für Wirtschaft und Arbeit (BMWA). Das Gesamtziel des Forschungsprojektes ist die Entwicklung eines universellen und mobilen Kommunikationsframework für die Hightech Branche. Das Framework ist Open Source und komplett in Java entwickelt. Es soll auch kleinen und mittelständigen Betrieben eine kostengünstige Lösung zur Automatisierung ihrer Produktion zur Verfügung stellen. Zum Zeitpunkt der Erstellung dieser Arbeit befindet sich das Projekt noch mitten in der Entwicklungsphase. Die Veröffentlichung einer ersten Version ist für Mitte 2005 geplant.

4.1 Architektur von Message to Anywhere

Abbildung 13 zeigt die Gesamtübersicht der M2A Middleware die sich aus folgenden drei Teilen zusammensetzt:

- Das Framework, welches für den physikalischen und logischen Transport der Daten zwischen den Datenquellen und Datensenken zuständig ist.
- Ein Kernstück ist ein generischer Adapter, der eine schnelle und sehr flexible Anbindung von Clienten an das Framework ermöglicht. M2A besitzt ein so genanntes Adaptertoolkit, so dass sich die Entwicklung eines speziellen Adapters für einen Client sehr einfach gestaltet. Der Adapter stellt die API zur Kommunikation mit dem Framework zu Verfügung. Weiterhin ist er für eine Umsetzung der Daten, die das Framework liefert, auf die verschiedensten Protokolle zuständig, die in der Industrie zum Einsatz kommen.
- Der Funktionsumfang des Frameworks kann mit Hilfe von Extensions erweitert werden. Es können hier beispielsweise Workflowsysteme integriert oder ein SMS-

Server angekoppelt werden. Auch Anwendungen zum Monitoring oder der Administration des Systems werden als Extensions verwirklicht. Auch der Webserver, der in dieser Arbeit entwickelt wurde, ist als eine solche Extension zu verstehen.

Die M2A Middleware setzt auf einem Applicationserver auf. Der aktuelle M2A-Prototyp verwendet die OpenSource Lösung JBOSS [JBoss04]. Es wird hier allerdings auch möglich sein andere Produkte einzusetzen.

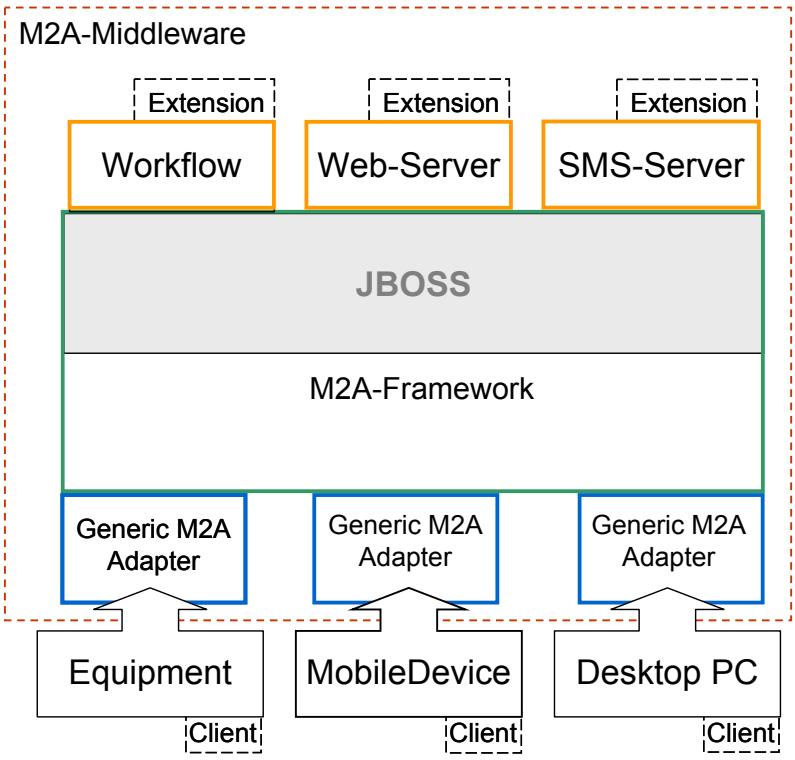


Abbildung 13: Gesamtübersicht über die M2A Middelware

Die Kommunikation in M2A ist Nachrichtenbasiert. Als Grundlage dient zur Zeit der Java Message Service (JMS). Ein M2A-Adapter stellt eine API zum Versenden und Empfangen von Messages zur Verfügung. Diese API erlaubt das Verwenden von verschiedenen Kommunikationsmodellen (Publish-Subscribe, Asynchrones Request-Response und Synchrones Request-Response).

4.2 Das Service Konzept von Message to Anywhere

Mit M2A soll die Integration von Maschinen (in der Halbleiterindustrie spricht man von so genannten Equipments) in eine Kommunikationsinfrastruktur einer Halbleiterfabrik wesentlich vereinfacht werden. Equipments können ihre Funktionen über das M2A Framework anbieten. Ein einfaches Beispiel hierfür ist Pumpe XY anschalten. In M2A sind diese Funktionen als so genannte Services verfügbar, und können von Clienten benutzt werden. Hierfür wird ein Konzept benötigt, das eine flexible Abbildung der Services eines Equipments ermöglicht.
Die Entwicklung eines solchen prototypischen Konzeptes in Zusammenarbeit mit dem Institut für Automatisierungs- und Softwaretechnik (IAS) der Universität Stuttgart ist ein Teil dieser Diplomarbeit. Das Entwickelte Konzept muss als Ansatz gesehen werden. Im endgültigen Release von M2A wird hier ein weitaus komplexerer Mechanismus eingeführt werden, der eine komplette Abbildung der hochkomplexen Funktionen von Equipments genügt. Im Vordergrund stand hier klar eine Lösung für Test- und Demonstrationszwecke.

Abbildung 14 zeigt die Struktur der Services auf einem Equipment. Ein RootService ist eine Funktionalität, die dem Benutzer direkt zur Verfügung steht. Man kann ihn als Einstiegspunkt auf einem Equipment sehen. Um einen RootService ausführen zu können, kann es nötig sein, dass Vorbedingungen erfüllt sein müssen. Mit anderen Worten: Weitere Services müssen zuvor ausgeführt werden bzw. sich in einem entsprechenden Zustand befinden. Nur wenn alle Services, die an einen RootService geknüpft sind, bereit sind, d.h. sich in einem entsprechenden Zustand befinden, kann dieser ausgeführt werden. Ein RootService ist von seinem Aufbau völlig identisch mit einem Service. Der einzige Unterschied besteht darin, das er auf einem Equipment nach außen hin sichtbar ist.

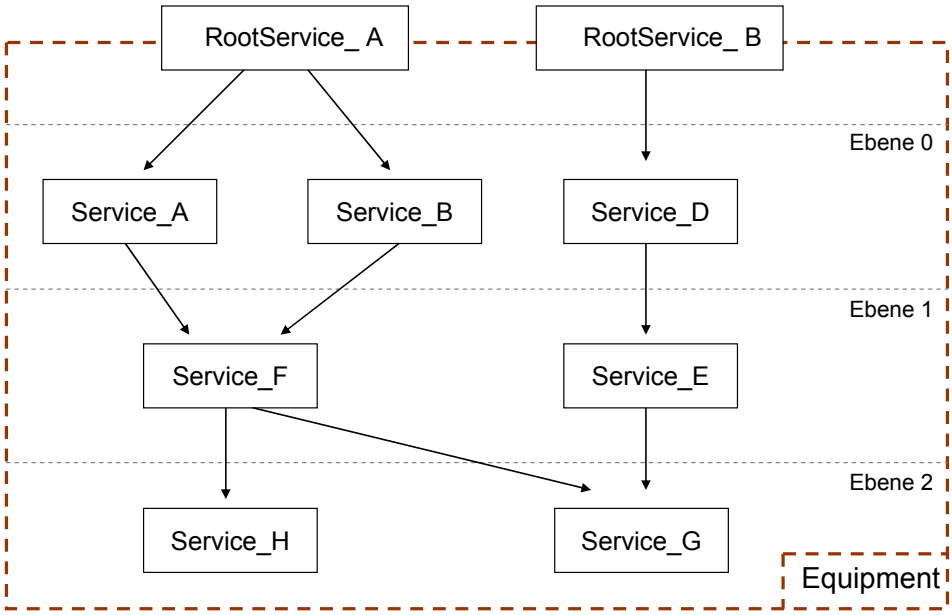


Abbildung 14: Aufbau der Services in einem Equipment

Um hier keine sehr umfangreiche Service Beschreibungssprache einführen zu müssen, wurde auf eine existierende Sprache zurückgegriffen, in der man sehr komplexe Zusammenhänge darstellen kann: Java.

In Abbildung 15 ist das Klassendiagramm der Servicestruktur zu sehen. Der Zugriff auf die Services erfolgt über das Interface `ServiceAccess`, das von einem Client implementiert werden muss. Auf diesem Client wird die Servicestruktur aufgebaut und er bekommt die Möglichkeit zur Ansteuerung der Services. Hierüber hinaus ist er für die Kommunikation mit M2A zuständig.

Mit der Methode `isServiceAvaiable` kann überprüft werden, ob alle Vorbedingungen für einen Service erfüllt wurden und ob er zur Ausführen bereit ist. Hierfür hält ein Service Referenzen auf seine Vorgänger (Predecessors) von denen die Ausführung dieses Service abhängt. Die Ausführung des Service erfolgt mit dem Aufruf der Methode `invokeService`, die beliebige Parameter erhalten kann. Um in Erfahrung zu bringen, welche Parameter die Ausführung eines Services erwartet, dient die Methode `getInvocationInfo`. Sie liefert ein Objekt von der Klasse `InvocationInfo`, das zum einen die Parameter selbst als Class Objects und zum anderen die zugehörigen Wertigkeiten dieser Parameter enthält. Ein Service, der als Parameter beispielsweise einen Integerwert erwartet, muss für den Anwender zusätzlich noch die Information über die

Wertigkeit der Eingaben bereithalten. Ein konkretes Beispiel währe ein Parameter „int 5“ der einer Wertigkeit von „5 bar“ entspricht.
Des weitern bietet das Interface `ServiceAccess` die Möglichkeit eine textuelle Beschreibung des Service abzufragen (`getDescription`) und alle RootServices (`getRootServices`) eines Clienten zu ermitteln. Das Interface `ServiceBuilder` definieret die Methoden um die Service Struktur auf dem Client aufzubauen.

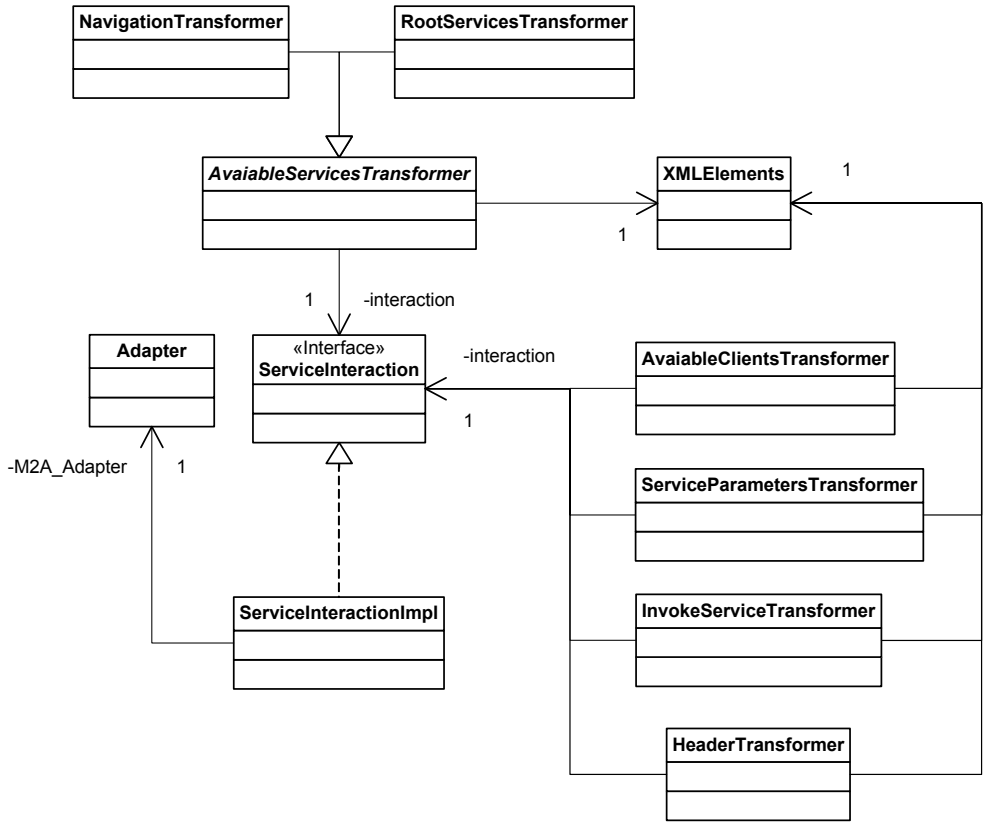


Abbildung 15: Klassendiagramm der Servicestruktur

Da die Kommunikation innerhalb von M2A Nachrichtenbasiert ist, muss der Client diese Nachrichten umsetzten und die entsprechende Methode auf den Services ausführen. Jede Methode, die das Interface `ServiceAccess` definiert, wird in einer speziellen M2A-Message abgebildet. Auch das Ergebnis wird in eine M2A-Message verpackt und an den Client zurückgesendet. Wenn eine Exception in einem Service auftritt, wird diese als Ergebnis zurückgesendet und kann vom anfragenden Client entsprechend verarbeitet werden.
Diese Service Struktur wird den verschiedensten Mobilen Endgeräten über einen Webserver zugänglich gemacht.

Praktische Umsetzung des Service Konzeptes

Mit der beschrieben Servicestruktur sollen die Funktionen einer 3D- Reinigungsanlage (Abbildung 16) auf Services abgebildet werden.

Bei dieser Anlage handelt es sich um einen weltweit einzigartigen Prototypen. Mit ihm ist es möglich Motorbauteile (z.B. Zylinder oder Kurbelwellen) automatisiert zu reinigen. Dies geschieht in der Automobilindustrie zurzeit noch in Handarbeit. Mit der automatisierten Reinigung wird versucht bessere Ergebnisse zu erreichen, da bei modernen Hochleistungsmotoren kleinste Verunreinigungen den Betrieb des Motors stören können. Die Anlage besteht aus einer hermetischen Kammer. Diese Kammer wird vor dem eigentlichen Reinigungsvorgang automatische gereinigt, und so eine Reinraumumgebung geschaffen.

Das zu reinigende Werkstück ist drehbar in der Kammer gelagert. Ein Roboter in der Kammer fährt das Werkstück ab und reinigt es mit einer Düse. Hier können wässerige und chemische Reinigungsmittel zum Einsatz kommen. Die verwendete Reinigungsflüssigkeit wird aufgefangen, filtriert und analysiert. Dadurch lassen sich Rückschlüsse auf die erzielte Sauberkeit des Werkstückes ziehen. Weiterhin besteht noch die Möglichkeit ein Werkstück in einem Ultraschallbecken, das in der Kammer integriert ist, zu reinigen.

Abbildung 16: 3D- Reinigungsanalge

Die Anlage wird über eine SPS (Speicherprogrammierbare Steuerung) gesteuert. Auf diese SPS kann mit einer Steuersoftware, die eine Java API besitzt, zugegriffen werden. Diese Software liefert weiterhin ein Steuerprogramm für die 3D-Reinigungsanlage. Anhand der GUI (Graphical User Interface) dieses Steuerptogramms und der Java API wurden vorhandene Funktionen in die Service-Struktur eines Clients übertragen und dieser in das M2A-Framework integriert.

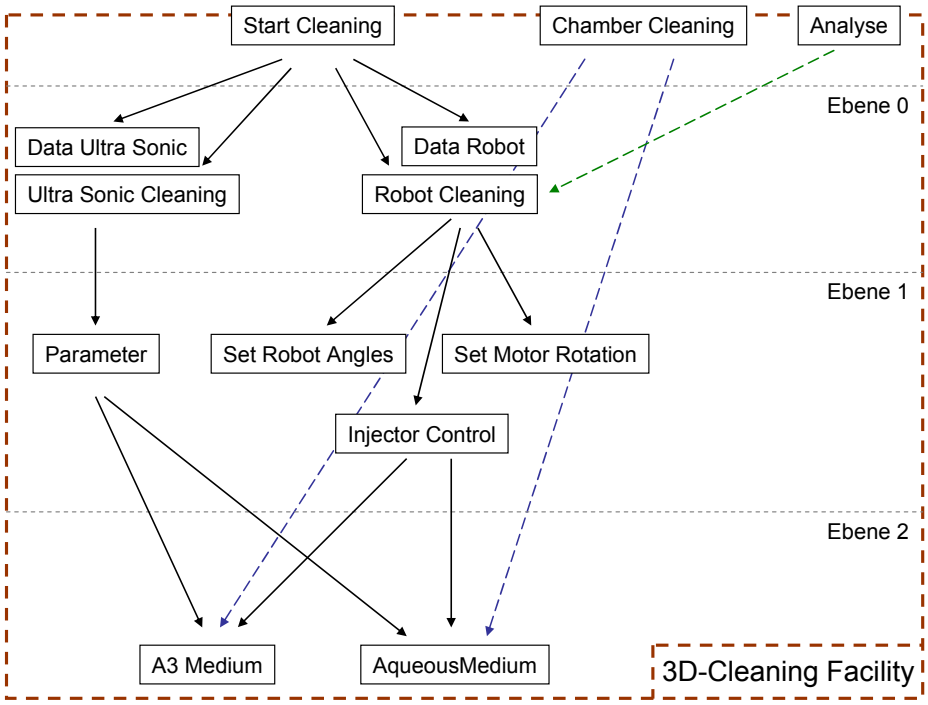


Abbildung 17: Service-Struktur der 3D-Reinigungsanlage

Die Funktionen der Anlage konnten alle auf die Service-Struktur abgebildet werden. Dadurch, dass die Maschine einen technischen Defekt hatte und der Hersteller der SPS-Steuersoftware die API-Dokumentation nicht rechtzeitig liefern konnte, war es zeitlich nicht mehr möglich, die Ansteuerung an der Maschine direkt zu testen. Die Services sind somit nur als Dummy implementiert worden.

5 Realisierung der Aufgabenstellung

5.1 Konzeption und Planung

In Zusammenarbeit mit dem M2A-Entwicklungsteam wurden Anwendungsfälle (Use-Cases) festgelegt, die dem Benutzer im aktuellen M2A-Prototyp zur Verfügung stehen. Grundsätzlich werden alle Use-Cases auf allen Endgeräten (PC, PDA, Smartphone, ...) über einen entsprechenden Browser (HTML oder WML) dem Anwender angeboten. Allerdings müssen die Use-Cases und die Darstellung abhängig von der Gerätegruppe adaptiert werden.
Abbildung 18 zeigt die möglichen Anwendungsfälle in einem Unified Modelling Language (UML) Use-Case Diagramm. In Tabelle 4 werden sie mit ihren jeweiligen Vorbedingungen beschrieben.

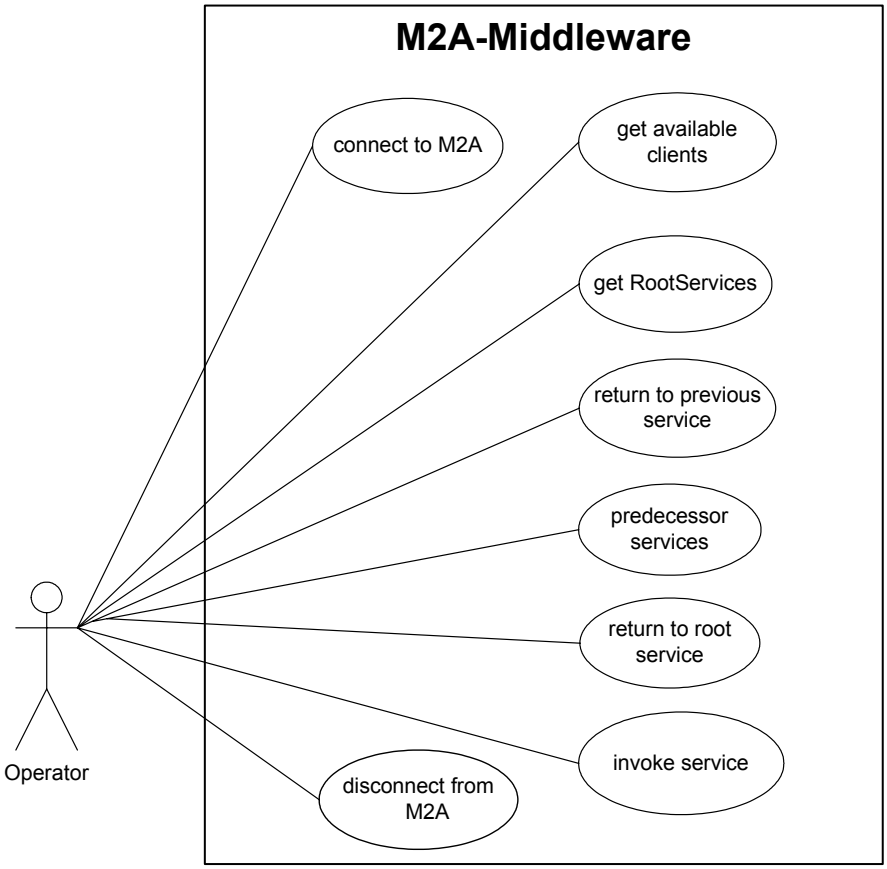


Abbildung 18: Use-Case Diagramm für die Interaktion mit Services

Use-Cases:	**Ziel:**	**Vorbedingungen**
connect to M2A	Verbindung mit dem M2A-Framework herstellen und Überprüfung der Benutzerdaten.	(a) Es muss eine HTTP-Verbindung zwischen Client und Webserver bestehen.
get available clients	Auflistung aller, für den Benutzer zugänglichen, Clienten.	(b) Die Verbindung zu M2A muss hergestellt sein. Zusätzliche Vorbedingung: (a).
get RootServices	Anzeigen aller RootServices auf einem Clienten.	(c) Der Benutzer muss zuvor einen Clienten ausgewählt haben. Zusätzliche Vorbedingungen: (a), (b).
predecessor services	Alle Vorgänger eines ausgewählten Services auflisten, d.h. eine Ebene tiefer im Service-Graphen navigieren.	(d) Der Service muss Vorgänger besitzen. Zusätzliche Vorbedingungen: (a),(b), (c).
return to root service	Auf allen Ebenen des Service Graphen kann eine Navigation zu dem zugehörigem RootService erfolgen.	(e) Man muss sich in der Service Struktur unterhalb des RootService befinden. Zusätzliche Vorbedingungen: (a), (b), (c), (d).
return to previous services	Es besteht die Möglichkeit innerhalb des Service-Graphen eine Ebene höher zu navigieren, bis der Einstiegs RootService erreicht ist.	(f) Der Benutzer darf sich in der Service Struktur nicht auf einem RootService befinden. Zusätzliche Vorbedingungen: (a), (b), (c), (d).
invoke services	Ausführen eines Services.	Der Service muss verfügbar sein, d.h. alle Vorbedingungen müssen erfüllt sein. Zusätzliche Vorbedingungen: (a), (b), (c).
disconnect from M2A	Verbindung mit dem M2A-Framework trennen.	Es muss eine Verbindung zu M2A bestehen. Zusätzliche Vorbedingung: (a).

Tabelle 4: Beschreibung aller Use-Cases

Im nächsten Schritt wurde ausgearbeitet, wie der Ablauf innerhalb des Webservers aussehen soll, und wie die einzelnen Use-Cases dem Benutzer dargestellt werden können. Abbildung 19 zeigt in einem Aktivitätsdiagramm den Ablauf innerhalb des Webservers und wie er auf verschiedene Benutzereingaben reagieren muss. Im Detail stellt sich dieser Ablauf folgendermaßen dar:

Da eine Anpassung der Daten auf das anfragende Gerät erfolgt, müssen zu Beginn Informationen über dieses Gerät ermittelt werden. Mit diesen Informationen kann nun die Ausgabe auf das entsprechende Gerät angepasst werden (Aktivität Abbildung 19, DatenAdaption). Bevor nun die eigentliche Applikation, nämlich das Ausführen von Services genutzt werden kann, muss der Benutzer sich authentifizieren, um eine Verbindung mit M2A herzustellen zu können. Im nächsten Schritt können alle Clienten abgefragt werden die dem Benutzer zugänglich sind. Nach der Auswahl eines Clienten werden die verfügbaren RootServices auf ihm angezeigt. Nun kann sich der Anwender einen RootService auswählen und hat die Möglichkeit durch den Service Graphen zu navigieren, einen verfügbaren Service auszuwählen und ihn auszuführen oder sich jederzeit vom M2A-Framework abzumelden.

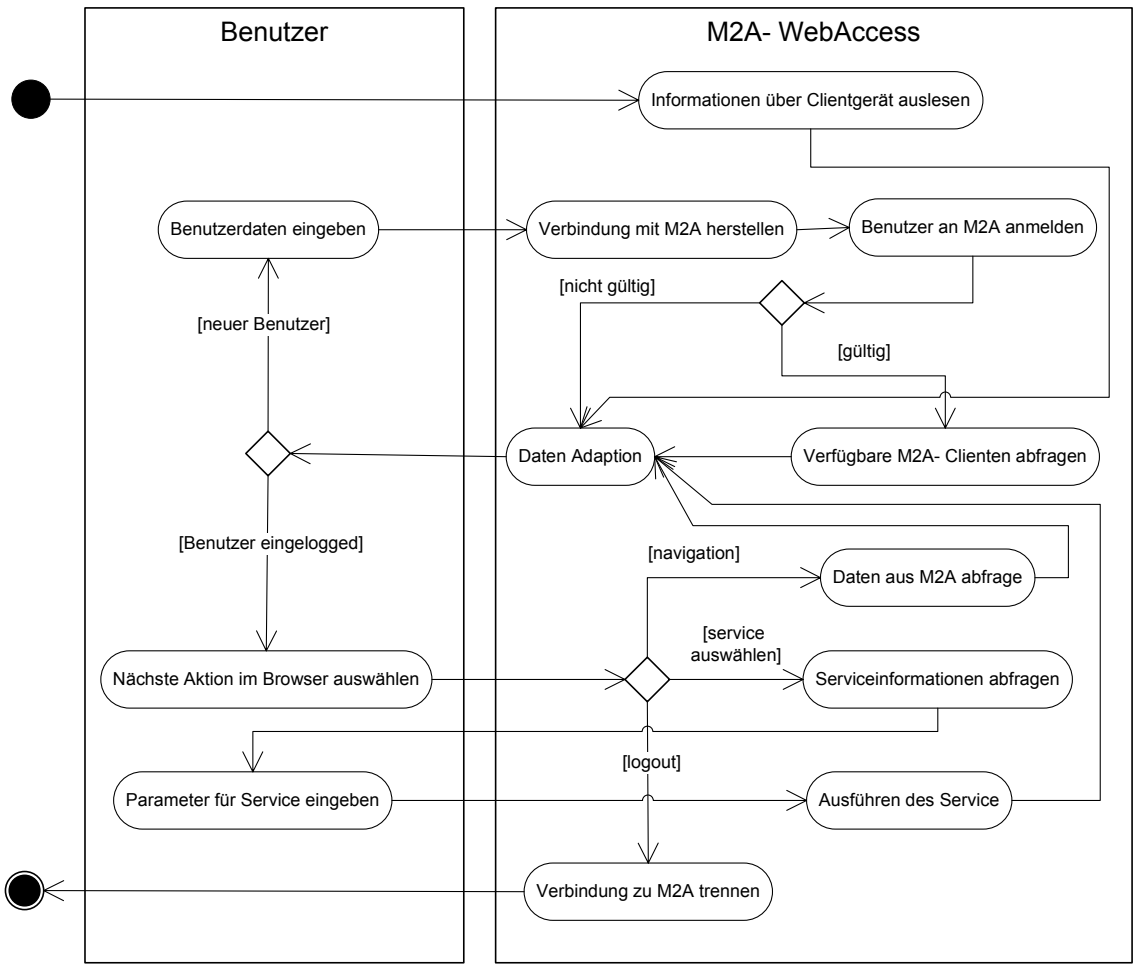


Abbildung 19: UML-Aktivitätsdiagramm; Ablauf innerhalb des Webservers

Der Webserver muss sich sehr gut in das M2A-Framework integrieren lassen. Da der aktuelle M2A-Prototyp auf einem JBOSS Applicationserver entwickelt wird, wurde der Webserver Apache Tomcat [Tomcat04] gewählt, da dieser direkt in JBOSS integriert ist. Cocoon wird als Servlet in diesem Tomcat Webserver betrieben.

In Abbildung 20 ist die Anordnung des Webservers in Verbindung mit dem M2A-Framework zu sehen. Da alle Clienten, also auch Mobile Endgeräte, über einen Adapter mit dem M2A-Framework verbunden werden, muss jede HTTP-Verbindung mit dem Webserver über einen Adapter auf M2A abgebildet werden. Jede HTTP-Session des Webserver muss, solange sie gültig ist, ihren eigenen Adapter zur Kommunikation mit M2A halten. Der Adapter stellt die API zum Versenden und Empfangen der M2A-Messages zur Verfügung. Die Kommunikation zwischen dem Webserver und einem Equipment erfolgt ausschließlich über M2A-Messages. Die Verbindung zum M2A-Framework wird zu Beginn einer HTTP-Session aufgebaut, und so lange gehalten bis diese nicht mehr gültig ist.

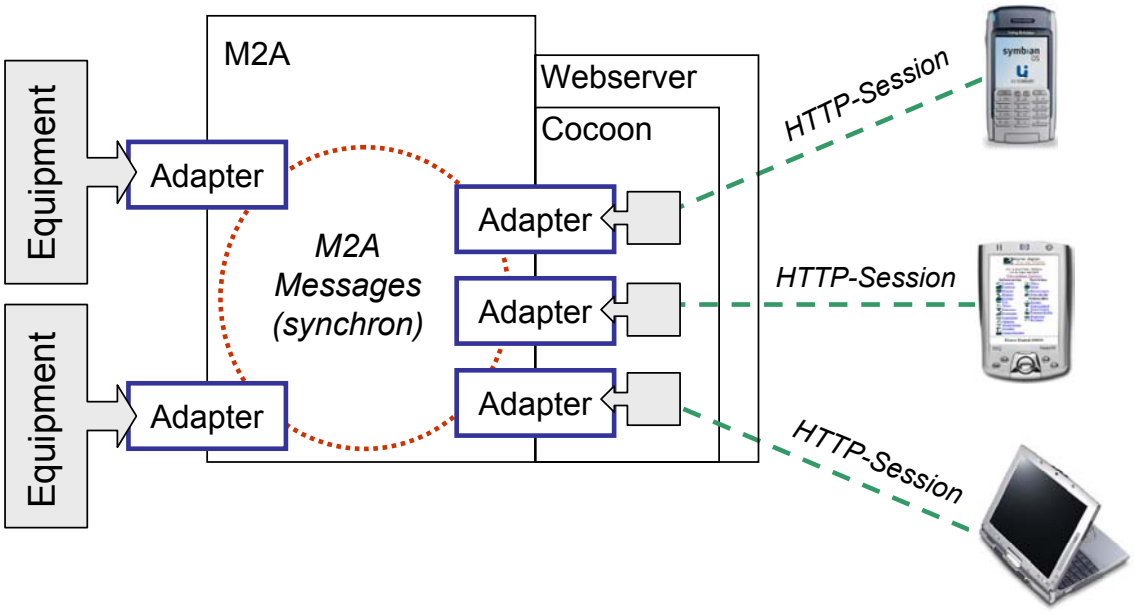


Abbildung 20: Anordnung von Webserver und M2A

Die M2A-Messages, die vom Adapter des Mobilen Endgerätes an den Adapter des Equipments geschickt werden, besitzen ein spezielles Format. Für jede Methode, die das Interface `ServiceAccess` (siehe Kapitel 4.2) definiert, wird eine entsprechende Nachricht

erstellt und versendet. Ein Equipment oder Client ist in der Lage diese Nachrichten auszuwerten, die entsprechende Aktion auszuführen und das Ergebnis zurückzusenden.
Da ein Webserver an den HTTP-Request-Response Zyklus gebunden ist, muss dies bei der Kommunikation mit dem M2A-Framework berücksichtigt werden. Aus diesem Grund erfolgt die Kommunikation zwischen Webserver bzw. dem Adapter des Clienten und M2A ausschließlich synchron. Dies bedeutet, dass der HTTP-Response zum Endgerät erst bei Erhalt einer Antwort auf die versendete M2A-Message erfolgen darf. Im Gegensatz zur asynchronen Kommunikation, wo zwar eine Antwort auf die versendete Nachricht erfolgen muss, aber der Client nicht auf diese wartet. Die Verbindung zwischen Adapter und M2A bleibt so lange geöffnet, wie eine HTTP-Session gültig ist.
Bei einer asynchronen Kommunikation können auch Nachrichten an den Adapter bzw. Client versendet werden, wenn dieser inaktiv ist. Im Fall des Webservers würde dies bedeuten, dass gerade kein Request erfolgt. Die asynchronen Nachrichten werden zwischengespeichert und dem Benutzer bei dem nächsten Request zur Verfügung gestellt. Somit könnte er über Zustandsänderungen auf dem Equipment informiert werden, d.h. genau genommen fordert er diese Informationen nur an, was keine akzeptable Lösung für zeitkritische Informationen (z.B. Alarmmeldungen) darstellt.
Diese Funktionalität konnte nicht implementiert werden, da der M2A-Prototyp noch nicht die nötigen Kommunikations-APIs auf dem Adapter hierfür bereitstellte.

Cocoon erlaubt eine strikte Trennung von Inhalten (Content), Ablaufsteuerung (Logik) und Präsentationsinformationen (Style) (siehe auch Kapitel 2.2, Abbildung 2). Diese strikte Separation of Concerns musste auch bei der Einwicklung des M2A-Webservers eingehalten werden. Sonst wären die Anforderungen, die das M2A-Projekt in Bezug auf Unterstützung verschiedener Ausgabeformate, einfacher Integration neuer Endgeräte und einer erweiterbaren und flexiblen Architektur stellt, nicht umsetzbar gewesen.
Abbildung 21 zeigt, wie das Prinzip der Separation of Concerns im M2A-Webserver umgesetzt wurde.
Inhalte (Content): Die gesamten Informationen/Daten, die M2A liefert werden in XML umgewandelt. Sie enthalten keinerlei Angaben über Formatierung oder Ausgabekanal (z.B. HTML, WML,...). In einer späteren Version von M2A wird mit Nachrichten gearbeitet, die direkt XML unterstützen, so dass eine Umwandlung nicht mehr nötig sein wird.

Ablaufsteuerung (Logik): Das Grundgerüst für die Logik wird in der Sitemap aufgebaut. Die Komponenten innerhalb der Sitemap manipulieren bzw. erzeugen die Daten für die Ausgabe. Durch das Hinzufügen neuer Pipelines oder ganzer Sitemaps und der Möglichkeit der Wiederverwendung und Eigenentwicklung von Komponenten ist das System flexibel und leicht erweiterbar.

Präsentationsinformationen (Style): Hier werden die Inhalte an das erforderliche Ausgabeformat angepasst. Für jedes Format steht ein XSLT-Stylesheet zur Verfügung. Dieser hat Zugriff auf das CC/PP Profil des Endgerätes und kann anhand dessen Informationen die Ausgabe entsprechend anpassen. Somit entfällt das Entwickeln eines Stylesheets für ein bestimmtes Endgerät, was wiederum zu einer Verringerung von redundanten Daten beiträgt und den Aufwand der Integration neuer Endgeräte erheblich minimiert.

Aus diesen Überlegungen wurde die genaue Architektur für die Implementierung entwickelt, die im Kapitel 5.2 vorgestellt wird.

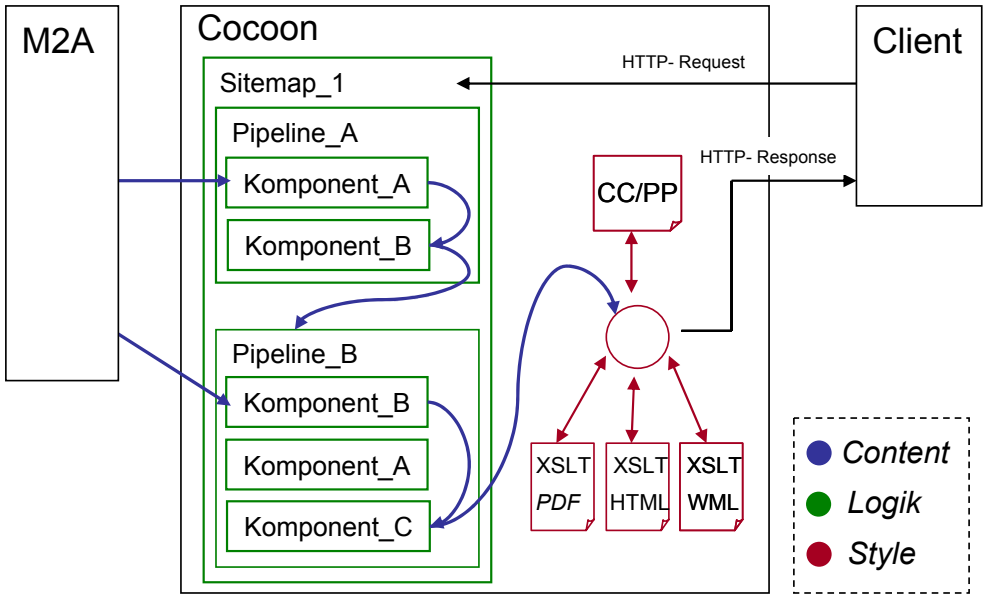


Abbildung 21: Umsetzung von Separation of Concerns im M2A-Webserver

5.2 Gesamtübersicht über die Implementierung

Die komplette Implementierung des M2A Webservers wurde nach Aufgabengebieten in folgende vier Pakete unterteilt:

ProfileDetection: Bei einem ersten Request eines Clients müssen zur Anpassung der Ausgabe Informationen über das anfragende Gerät gewonnen werden. Hierzu muss das Gerät erkannt und die nötigen Profildaten ausgelesen werden.

Authentication: Hier wird die Verbindung zum M2A-Framework hergestellt. Ein Benutzer muss sich gegenüber M2A mit seinem Benutzernamen und Passwort authentifizieren. Des Weiteren findet die Überprüfung auf eine gültige HTTP-Session und das Trennen einer bestehenden Verbindung zu M2A in diesem Paket statt.

Customization: Vor der Response des Servers muss eine Anpassung der Ausgabe auf das anfragende Gerät erfolgen. Hier werden die Informationen verwendet, die im Paket ProfileDetection gewonnen wurden.

ServiceInteraction: Dieses Paket stellt die eigentliche Anwendung dar. Nachdem ein Equipment vom Benutzer ausgewählt wurde, findet die gesamte Interaktion mit den Services in diesem Paket statt.

In Abbildung 22 ist die schematische Anordnung der einzelnen Pakete innerhalb von Cocoon dargestellt. Jeder Request eines Clients wird zunächst von der MainSitemap verarbeitet. Hier werden die Profildaten des anfragenden Client ermittelt (ProfileDetection), eine Verbindung mit dem M2A-Framework hergestellt, und die Benutzerdaten abgefragt (Authentication).

Die eigentliche Anwendung (ServiceInteraction) ist komplett in der ServiceSitemap untergebracht. Diese wird in die MainSitemap gemountet. In Cocoon wird eine solche Sitemap als SubSitemap bezeichnet. Alle Daten, die als Response an den Client geschickt werden, müssen zuvor auf das anfragende Gerät angepasst werden (Customization). Dies geschieht in den Sitemaps mit einem Mechanismus, der es erlaubt häufig verwendete Funktionalitäten zu gruppieren und wieder zu verwenden, der so genannten Resource. Dies wird im Kapitel 5.3.3, im Paket Customization, genauer beschrieben.

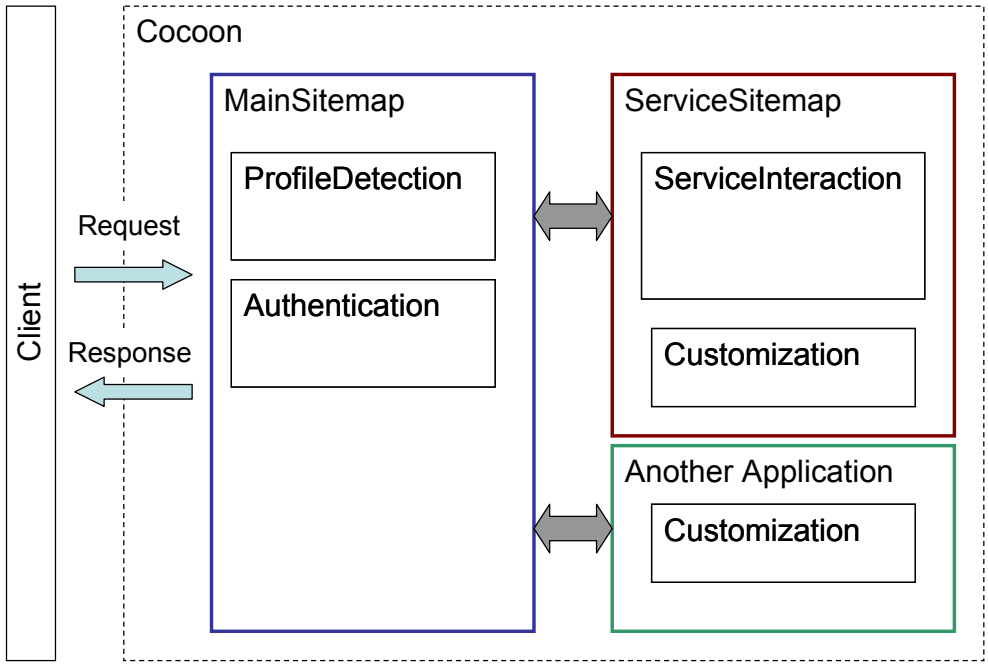


Abbildung 22: Schematische Anordnung der Pakete innerhalb der Sitemaps von Cocoon

Da alle Daten, die die eigentliche Anwendung (ServiceInteraction) betreffen, in einer SubSitemap zusammengefasst sind, kann man sehr leicht weitere Anwendungen in dieses Design integrieren. Eine neue Anwendung wird in einer weitere SubSitemap erstellt und in die MainSitemap eingemountet. Als Beispiel hierfür wäre eine Administrationskonsole anzuführen. Die Pakete ProfileDetection, Authentication und Customization können praktisch unverändert wieder verwendet werden.

Da es sich bei HTTP um ein zustandsloses Protokoll handelt, musste eine Lösung gefunden werden wie Informationen über mehrere Request-Response Zyklen, einer HTTP-Session, einem Benutzer zugeordnet werden können. Man spricht hier vom so genannten Session Tracking. Cocoon bietet hierfür einen Mechanismus an, um auf das Session Objekt eines Request zuzugreifen. Unter der Decke wird hier auf die Servlet Session Tracking API des Java SDKs (Software Development Kit) zurückgegriffen. Diese API abstrahiert die Mechanismen des Session Trackings wie beispielsweise das Verwenden von Cookies oder URL Rewriting. Beliebige Informationen können als Attribute an ein Session Objekt übergeben werden, und stehen so bei einem nächsten Request des Client wieder zur Verfügung.

Wie man das Session Objekt in Cocoon erhält und benutzt, wurde bereits in Kapitel 3.5.3, Beispiel 18 gezeigt.
Folgende Informationen müssen als Attribute in dem Session Objekt gespeichert werden:

- Benutzername.
- Adapter und entsprechende M2A-Session für die Kommunikation mit M2A.
- Informationen über die Eigenschaften des Clients, dem die Session zugeordnet ist. Dies ist das entsprechendes CC/PP Profil und ein weiteres Profil das im Kapitel 5.3.2 beschrieben wird.
- Für die eigentliche Anwendung, das Ausführen von Services (Kapitel 5.3.4, ServiceInteraction), wird zusätzlich der Navigationspfad des Benutzers durch den Service-Graphen eines Equipments gespeichert. Weiterhin werden alle Services markiert, die der Benutzer schon ausgeführt hat.

Wann und wie diese Daten in die Session geschrieben werden und wann sie benutzt werden, wird im nächsten Kapitel gezeigt, in dem die Implementierung im Detail vorgestellt wird.

5.3 Die einzelnen Pakete im Detail

In diesem Kapitel werden nun die vier Pakete der Implementierung im Detail beschrieben. Es werden keine kompletten Quellcodebeispiele aufgezeigt, da allein der in Java implementierte Teil einen Umfang von ca. 2000 Zeilen besitzt. Aus den einzelnen Paketen wurden die interessantesten Punkte herausgesucht, und werde in diesem Kapitel detailliert beleuchten.

5.3.1 ProfileDetection

Um eine angepasste Darstellung von Informationen auf unterschiedlichen Geräten zu erreichen, ist es zunächst nötig mehr über die Eigenschaften des anfragenden Gerätes in Erfahrung zu bringen. Dies gestaltet sich bei der großen Zahl von möglichen Endgeräten sehr schwierig (von einem einfachen Mobiltelefon über ein Smartphone, PDA bis hin zu einem Notebook). Eine Möglichkeit, diese Fülle unterschiedlicher Geräte und ihre Eigenschaften (Displaygröße, Übertragungsmedien, Eingabemöglichkeiten,...) zu beschreiben, wurde im Kapitel 3.3 mit dem CC/PP Profilen vorgestellt. Wie hier schon erwähnt wurde, ist zurzeit keine Übertragung dieser Profildaten von einem Client bzw. Browser zum Server möglich.
Das Problem besteht daraus, einen Request auf das passende Profil abzubilden. Die Informationen aus dem HTTP-Header erlauben keine eindeutige Identifizierung der anfragenden Geräte.
Die direkte Zuordnung eines Requests zu einem Endgerät und dadurch zu einem passenden CC/PP Profil ist nicht möglich.
Aus diesem Grund wurde in dieser Arbeit einen indirekten Weg gewählt, der nicht auf das anfragende Gerät selbst, sondern über den anfragenden Browser Rückschlüsse auf das Gerät bzw. die Gerätegruppe zieht. Dies geschieht über das Feld `user-agent` im Header des HTTP-Protokolls. Jeder Browser besitzt einen eindeutigen User-Agent-String, mit dem er sich gegenüber einem Server identifizieren kann.
Beispiel 20 zeigt den User-Agent-String des Netscape7 (a) im Vergleich mit dem Microsoft Internet Explorer für Windows CE (b).

```
a)Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b)Gecko/)...
b)Mozilla/4.0 (compatible; MSIE 4.01; Windows CE; PPC; 240x320)...
```

Beispiel 20: Zwei User-Agent-Strings im Vergleich (Auszug)

Das Problem hierbei ist, dass ein Browser nicht an die Hardware gebunden ist. Ein Mobiltelefon ist hier die Ausnahme, da hier der Browser meist fest an das Gerät integriert ist. Bei einer Anfrage von einem PDA ist dies schon schwieriger. Zwar werden hier Browser eingesetzt, die auf die Geräte zugeschnitten sind und über ihre User-Agent-Strings zu identifizieren sind, allerdings kann man nur bedingt auf die Hardware schließen. Ein

Microsoft Internet Explorer für Windows CE kann beispielsweise sowohl auf einem Gerät mit einer Bluetoothschnittstelle, als auch auf einem das kein Bluetooth unterstützt eingesetzt werden. Auf der anderen Seite sind beispielsweise bei PDAs die Bildschirmabmessungen (ca. 240 X 320 Pixel) relativ gleich, so dass diese Informationen wieder verwendet werden können. Eine Gerätegruppe, bei denen es nicht gelingt, mit Hilfe des User-Agent-Strings Rückschlüsse auf ihre Hardware zu ziehen sind Notebook und Tablet PC. Dies ist jedoch akzeptabel, da man bei dieser Gruppe davon ausgehen kann, dass sie in den Ein- Ausgabemöglichkeiten und Darstellung der Daten nicht eingeschränkt sind.

Die Abbildung von einem User-Agent-String auf die entsprechenden Profildaten geschieht in einer XML-Datei deren Aufbau in Beispiel 21 zu sehen ist und nun näher erläutert wird.
Für die Beschreibung der Geräteeigenschaften wird das CC/PP Profil verwendet. Ein Profil ist entweder einem Gerät zugeordnet oder bei mehreren ähnlichen Geräten, könne diese gruppiert werden und alle verwenden dasselbe Profil.
Wie im Beispiel zu sehen, kann man in dem Element `<device>`, das auf das CC/PP Profil verweist, 1...n verschiedene `<userAgent>` Elemente angeben, die wiederum einen User-Agent-String darstellen. Das Attribute `type` legt das Standard Ausgabeformat für diese Profile fest (`HTML` oder `WML`).
Wie sich im Laufe der Arbeit herausstellte, reicht die Angabe aus dem CC/PP Profil nicht aus, um eine möglichst flexible Anpassung und Darstellung der Informationen auf das Endgerät zu gewährleisten. Das Aufstellen von Regeln, die aus den Daten des Geräteprofils Rückschlüsse ziehen, auf den Umfang der Daten, die dargestellt werden sollen, erweist sich bei der Fülle von verschiedenen Endgeräten als zu unflexibel und sehr kompliziert. Die Angaben aus dem CC/PP Profil werden nur für eine angepasste Darstellung genutzt, nicht aber für den Umfang der Daten.
Eine zweite, einfache Beschreibungsmöglichkeit für den Umfang der darzustellenden Informationen wurde aus diesem Grund eingeführt, ein so genanntes ContentProfile. In ihm wird festgelegt, welche Informationen angezeigt und welche ausgespart werden. Dieses Profil ist speziell für die eigentliche Anwendung, das Ausführen von Services (Kapitel 5.3.4), zugeschnitten. Wenn eine weitere Anwendung auf dem Webserver realisiert werden soll, kann es Sinn machen hierfür ein eigenes ContentProfile zu erstellen. Hierdurch können einzelne Anwendungen unabhängig von einander arbeiten.

Welche Inhalte bei der Anzeige konkret ausgespart werden können, wird im Kapitel 5.3.3, in dem das Paket Customization vorgestellt wird, beschrieben. Das Element `<contentDescription>` verweist auf das entsprechende ContentProfile, das eine einfache XML-Struktur besitzt. Es werden einfach alle Elemente bzw. Inhalte hier eingetragen, die nicht angezeigt werden sollen.

Das Profil ist für alle `<userAgent>` Elemente gültig, die von dem `<contentDescription>` Kindelement sind. Somit wurde eine Trennung zwischen Profildaten, die für die Adaption der Anzeige auf der Präsentations-Ebene (CC/PP Profil) und Inhaltlicher-Ebene (ContentProfile) vorgenommen.

Die Struktur der XML-Datei wurde in einer einfachen DTD definiert. Dadurch wurde die Datei validierbar und eventuelle Fehler bei der Erweiterung sind leichter erkennbar.

```
<devices>
  ...
  <!-- ContentProfile, gueltig fuer userAgent Kindelemente -->
  <contentDescription src="pdaContentProfile.xml">
    <!-- CC/PP Profil, gueltig fuer userAgent Kindelemente -->
    <device src="hp_compaq_iPAQ.rdf" type="HTML">
     <userAgent name="Mozilla/4.0 (compatible; MSI..."/>
     <userAgent name="Mozilla/4.76 [en] [PalmOS; U; WebPro..."/>
    </device>
   </contentDescription>

   <contentDescription src="wapHandyNewGenerationProfile.xml">
    <device src="GX10.rdf" type="WML">
      <userAgent name="SHARP-TQ-GX10 v0.1 ..."/>
    </device>

    <device src="T610R101.rdf" type="WML">
      <userAgent name="SonyEricssonT610/R101 Profile..."/>
    </device>
  </contentDescription>
   ...
</devices>
```

Beispiel 21: Auflösung des User-Agent-Strings zu verschiedene Profilen

Für dieses Paket wurden zwei Komponenten erstellt (Abbildung 23). Die `ProfilerAction` ist eine Erweiterung der Cocoon Action Komponente. Der prinzipielle Aufbau einer solchen Action Komponente wurde in Kapitel 3.5.3, Beispiel 18 gezeigt. Die `ProfilerAction` prüft, ob in der Session des Request schon ein Geräteprofil hinterlegt ist.

Wenn dies der Fall ist, kann die Verarbeitung hier abgebrochen werden. Ist jedoch noch kein Geräteprofil in der Session hinterlegt, handelt es sich um eine neue Session, in der die Profildaten noch gespeichert werden müssen. Die Action ermittelt aus dem HTTP-Request-Header den User-Agent-String des Browsers (Schritt eins). Anschließend wird über den Service Manager die selbst entwickelte Avalon Komponente `ProfilesReader` instantiiert, und dieser der User-Agent-String übergeben. Der `ProfilesReader` kann nun, wie Anfangs beschrieben, über den User-Agent-String und der Konfigurationsdatei für die einzelnen Profile, das entsprechende CC/PP Profil und das zugehörige ContentProfile ermitteln (Punkt zwei). Beide Profile werden mit einem DOM-Parser ausgelesen. Die Daten des CC/PP Profils (Elementname und Wert) werden in eine Java `Map` geschrieben. Die Content Beschreibung wird in eine `ArrayList` geschrieben. Beide werden als Attribute in der Session des aktuellen Requests gespeichert (Punkt drei). Hierdurch hat man, solange diese Session gültig ist, in jeder Komponente innerhalb von Cocoon Zugriff auf diese Daten. Zusätzlich wird der Wert des Attributs `type`, des Elements `device` aus der Konfigurationsdatei in Beispiel 21, als Attribut in der Session gespeichert. Diese Information benötigt das Paket Customization (Kapitel 5.3.3), um das entsprechende Ausgabeformat zu generieren.

Weiterhin werden die Pfadangaben zu den Verzeichnissen der Profildaten in einer Java-Properties Datei angegeben. Die Komponente ProfilerAction kann bei ihrem Aufruf in der Sitemap die Pfadangaben aus der Properties Datei auslesen. Cocoon stellt hierfür schon einen sehr komfortablen Mechanismus bereit, der hier auch eingesetzt wurde. Im Kapitel 5.3.2, Beispiel 22 wird der Einsatz dieses Paketes im Zusammenspiel mit dem Paket Authentication innerhalb der Sitemap gezeigt.

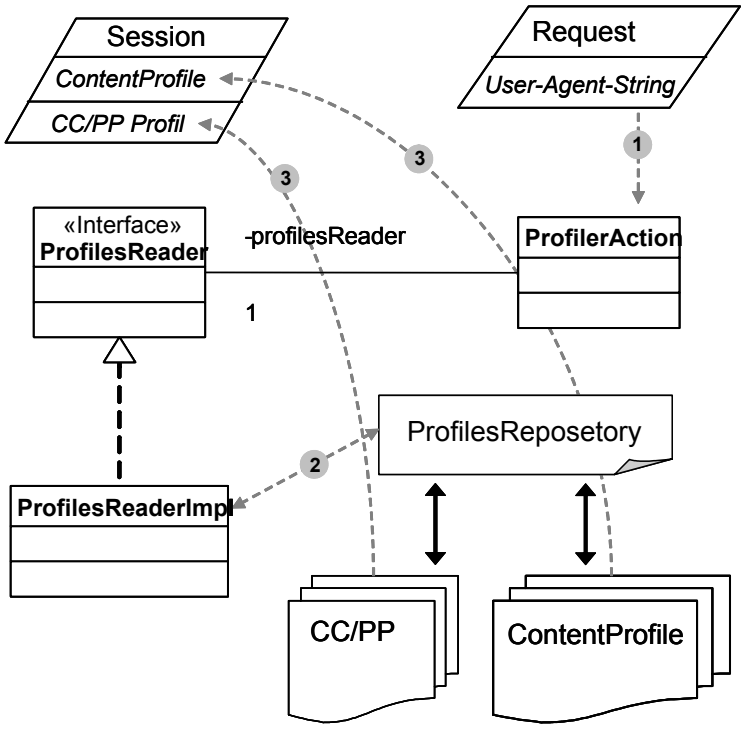


Abbildung 23: UML Klassendiagramm mit dem Ablauf der Profilverarbeitung

5.3.2 Authentication

Dieses Paket muss drei Aufgaben erfüllen. Es muss zum Ersten über einen M2A-Adapter die Verbindung zum M2A-Framework herstellen und den Benutzer anmelden. Ist der Benutzer korrekt mit M2A verbunden, muss zum Zweiten bei jedem Request des Browsers überprüft werden, ob die zugehörige HTTP-Session noch gültig ist. Die dritte Aufgabe ist es, eine bestehende Verbindung zu M2A wieder zu trennen. Für jede Aufgabe wurde eine Action Komponente (vergleiche Kapitel 3.5.3, Beispiel 18) entwickelt, die im Folgenden nun vorgestellt werden.

LoginAction: Die Hauptaufgabe besteht darin, eine Verbindung zum M2A-Framework über einen Adapter herzustellen und den Benutzer am Framework anzumelden.
Abbildung 24 zeigt das UML Sequenzdiagramm vom Aufbau einer Verbindung und dem Senden einer Nachricht über den M2A-Adapter. Zu Beginn muss über eine Adapterfactory ein Adapter instantiiert werden. Das Erzeugen einer M2A-Session erfolgt auf diesem Adapter mit der Methode `createSession`. Hier können Sessions für verschiedene

Kommunikationsarten (Publish-Subscribe, Asynchronen Request-Response und Synchrones Request-Response) generiert werden.

In dieser Arbeit wird, wie in Kapitel 5.1 schon erwähnt, ausschließlich die Synchrone Kommunikation (Synchrones Request-Response) benutzt. Bei einer Anfrage an das Framework wartet der aktuelle Thread bis er eine Antwort erhalten hat bzw. ein Timeout erreicht ist. Das synchrone Senden einer Nachricht erfolgt mit der Methode `request`, die als Rückgabewert die entsprechende Antwortnachricht liefert.

Der Adapter und die M2A-Session werden zusammen mit den eingegebenen Benutzerdaten im Session Objekt als Attribute gespeichert. Bei jedem Request des Clients kann über dieses Session Objekt auf den M2A-Adapter bzw. die M2A-Session zugegriffen werden und eine Kommunikation mit dem M2A-Framework erfolgen.

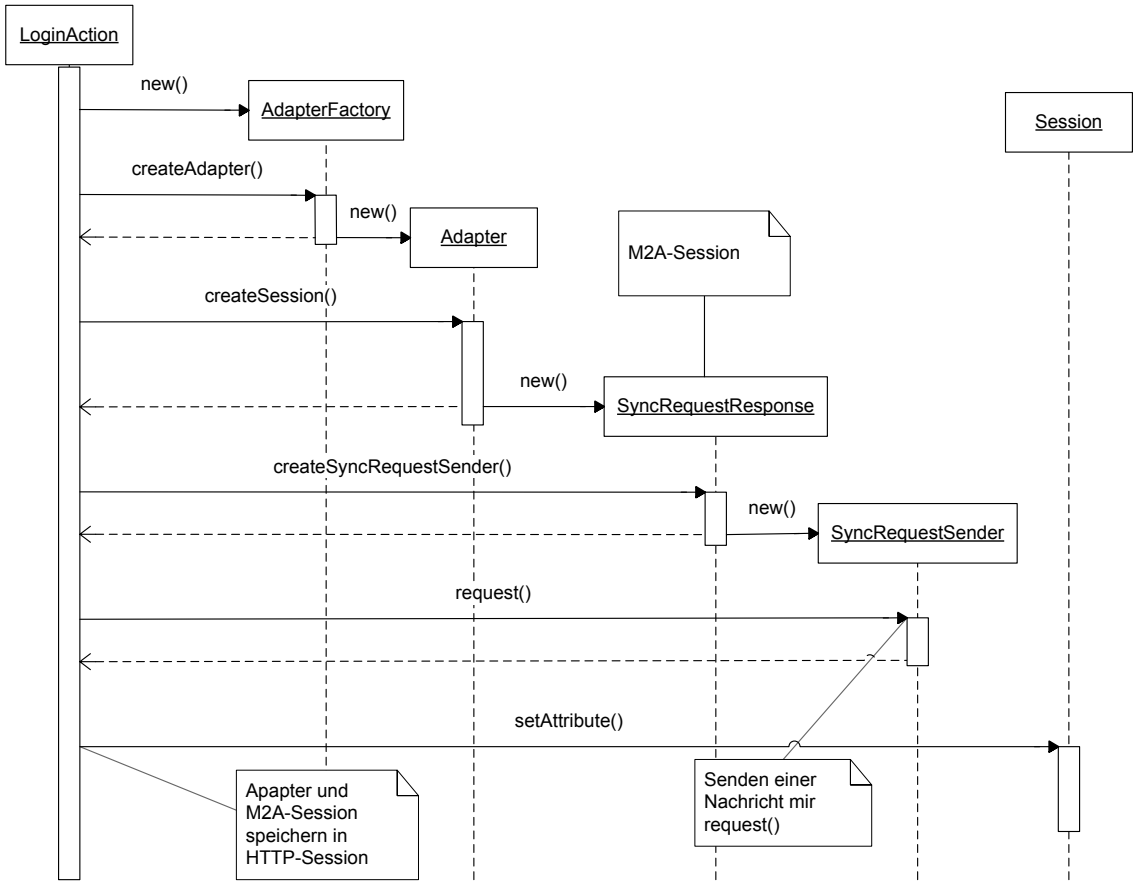


Abbildung 24: Kommunikation mit M2A über den Adapter und dessen Messaging-API

SessionValidatorAction: Bei jedem Request muss überprüft werden, ob die zugehörige HTTP-Session noch gültig ist. Wenn das Timeout der Session (maximale Zeit die der Benutzer keinen neuen Request sendet) erreicht ist, ist diese nicht mehr gültig. Wenn dies der Fall ist, wird die Verarbeitung dieser Action Komponente abgebrochen. Das Trennen einer Verbindung zum M2A-Framework geschieht in der LogoutAction.

LogoutAction: Wenn ein Benutzer sich von M2A abmelden möchte, oder wenn die HTTP-Session des Requestes ungültig ist, wird diese Action aufgerufen. Wenn eine Verbindung zu M2A bestand, wird der Benutzer abgemeldet und die Verbindung auf dem Adapter, sowie der Adapter selbst werden geschlossen.

Im Beispiel 22 ist ein Auszug aus der Sitemap zu sehen. Hier wird das Zusammenspiel der Geräteerkennung (Paket ProfileDetection, Kapitel 5.3.1) und diesem Paket mit den `Login-SesionValidatorAction-` und `LogoutAction` gezeigt.
Alle Requests werden in Zeile vier in den Abschnitt `protected` (Zeile 37) weitergeleitet. Hier wird der Request von der `SessionValidatorAction` (`sessionValid`) überprüft. Verfügt der aktuelle Request schon über eine gültige Session, gelangt er in den geschützten Bereich. Wird die Verarbeitung der Action jedoch abgebrochen, handelt es sich entweder um einen neuen Request oder das Timeout der Session ist abgelaufen. In beiden Fällen erfolgt eine Umleitung in den Abschnitt `logout` (Zeile 45). Hier werden durch die `LogoutAction` eventuell bestehende Verbindungen zum M2A-Framework getrennt und der Request umgeleitet zum Abschnitt `login` (Zeile acht). Bei einem ersten Request des Clients erfolgt zunächst die Geräterkennung über die Action `deviceProfiler` (Profile Detection Kapitel 5.3.1). In jedem Fall wird aber die Eingabemaske für die Benutzerdaten generiert und die Resource `outFormatTransformation` gerufen, in der das Paket Customuzation (Kapitel 5.3.3) die Aufbereitung der Daten für das jeweilige Endgerät vornimmt.
Hat nun der Benutzer seine Daten (Benutzername und Passwort) eingegeben, erfolgt die Anmeldung über den Abschnitt `dologin` (Zeile 25. Wenn eine Verbindung mit dem M2A-Framework hergestellt werden konnte, d.h. die Verarbeitung der Action `m2aLogin` wurde nicht abgebrochen, gelangt der Request wieder in den Abschnitt `protected`. Hier kann

der Benutzer nun den geschützten Bereich betreten und auf dem Webserver arbeiten. Ist dies nicht der Fall, erfolgt die Umleitung zu `logout`. Der Gesamte Zyklus beginnt von neuem.

```
<map:pipeline>
<!-- Einstiegspunkt für alle Requests -->
<map:match pattern="" type="wildcard">
   <map:redirect-to uri="protected"/>
</map:match>

<!-- Abfragen der Profildaten und Anzeige der Login- Maske -->
<map:match pattern="login" type="wildcard">
  <map:act type="deviceProfiler">
    <map:parameter name="path_deviceProfiles"
                   value="{mtoa-prop:path_deviceProfiles}"/>
          <map:parameter name="path_contentProfiles"
                         value="{mtoa-prop:path_contentProfiles}"/>
          <map:parameter name="profilesLocation"
                         value="{mtoa-prop:profilesLocation}"/>
          <map:parameter name="httpSessionTimeout"
                         value="{mtoa-prop:session-timeout}"/>
       </map:act>

       <map:generate src="data/xml/login.xml"/>
       <map:call resource="outFormatTransformation"/>
</map:match>

<!--Verbindung zu M2A herstellen und ausführen des Logins -->
<map:match pattern="doLogin">
  <map:act type="m2aLogIn">
      <map:parameter name="userName"
                     value="{request-param:userName}"/>
      <map:parameter name="passWord"
                     value=="{request-param:userName}"/>
    <map:redirect-to uri="protected"/>
   </map:act>
    <map:redirect-to uri="logout"/>
</map:match>

<!-- Geschuetzter Bereich -->
<map:match pattern="protected">
  <map:act type="sessionValid">
     <map:redirect-to uri="data/CLIENT"/>
  </map:act>
    <map:redirect-to uri="logout"/>
</map:match>

<!-- Verbindung zum M2A Framework wieder trennen , wenn noetig -->
<map:match pattern="logout" type="wildcard">
   <map:act type="m2aLogOut"/>
   <map:redirect-to uri="login"/>
</map:match>
</map:pipeline>
```

Beispiel 22: Login- und SessionValidatorAction im Zusammenspiel mit der ProfileDetection

5.3.3 Customization

Die Anpassung der Daten für den Client muss vor jeder Response erfolgen. Um hier keinen redundanten Code zu erzeugen, wurde auf einen speziellen Mechanismus innerhalb der Sitemap zurückgegriffen, die Resource. Diese ist eine Pipeline, die man in einer Sitemap beliebig oft wieder verwenden kann. Sie kann alle Komponenten enthalten, die auch in einer Pipeline vorkommen dürfen. Eine Resource wird in der Sitemap einmal definiert und bekommt einen eindeutigen Namen zugewiesen. Über diesen Namen kann man mit Hilfe des `<map:call resource="..."/>` Befehls (vergleiche Beispiel 22, Zeile 21) den Request bzw. den SAX-Eventstream aus einer beliebigen Pipeline in die Resource umleiten und die Verarbeitung dort fortsetzen.
In Abbildung 26 ist der Ablauf der Anpassung der Daten vor der Response an den Client zu sehen. Die Sitemap Resource erhält den SAX-Eventstream mit den Daten aus einer vorherigen Pipeline (beispielsweise Aufruf eines Services oder Abfrage aller RootServices im Paket ServiceInteraction). Die Daten werden nun in zwei Stufen an die Eigenschaften des Client angepasst.
Zunächst werden in dem **ContentTransformer** diejenigen Elemente aus dem SAX-Eventstream herausgefiltert, die im ContentProfile angegeben wurden, das wie das CC/PP Profil in der Session des Requests gespeichert ist (Kapitel 5.3.1, ProfileDetection). Der ContentTransformer ist recht einfach aufgebaut. Als abstrakte Basisklasse dient der `AbstractSAXTransformer`, dessen Grundgerüst in Kapitel 5.3.4, Beispiel 25 zu sehen ist. Der SAX-Eventstream wird von diesem Transformer analysiert und diejenigen Elemente herausgenommen, die im ContentProfile angegeben wurden. Die entsprechenden Informationen werden somit nicht mehr angezeigt.
Abbildung 25 zeigt die Seite für die Auswahl von Services auf einem Standard HTML-Browser. Alle Bereiche der Seite die rot gekennzeichnet sind, können im ContentProfil angegeben werden und somit für bestimmte Gerätegruppen ausgeblendet werden. Daneben ist die Seite auf einem WAP-Browser in einem einfachen Mobiltelefon zu sehen. Obwohl hier möglichst viele Informationen ausgespart wurden, sind die Darstellungsmöglichkeiten auf diesem Gerät so eingeschränkt, dass eine Bedienung nur noch theoretisch möglich ist und diese einem Benutzer schwer zugemutet werden kann.

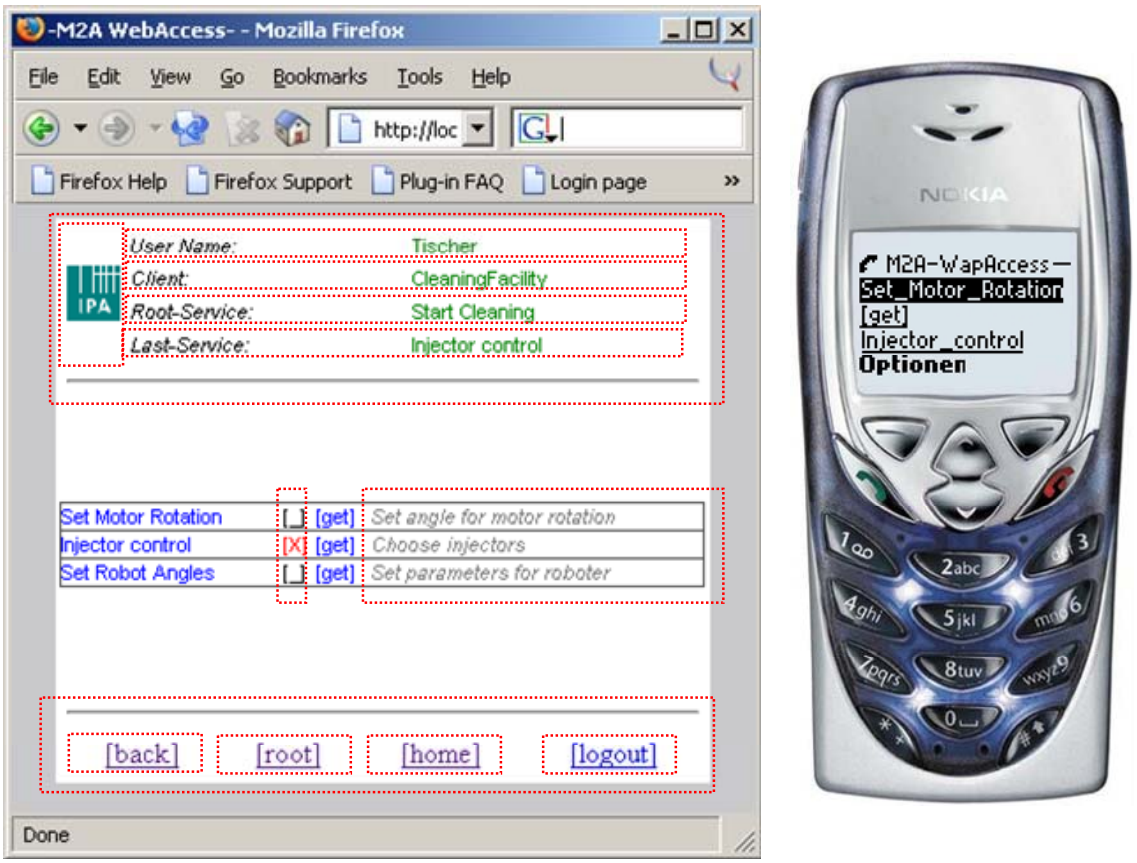


Abbildung 25: Uneingeschränkte Anzeige und eine extrem eingeschränkte Anzeige

In einem zweiten Schritt werden die Daten, die immer noch als XML bzw. als SAX-Eventstream vorliegen, in das Ausgabeformat (z.Zt. HTML oder WML) umgewandelt. Dies geschieht mit dem **PresentationTransformer.** Dieser Transformer ist eine Erweiterung des Standard XSLT-Transformer von Cocoon. Man kann ihm einen XSLT-Stylesheet angeben und hiermit die XML-Daten des SAX-Eventstreams in ein beliebiges Format umwandeln. Um nun eine, auf das Endgerät angepasste Transformation zu erreichen, muss das XSLT-Stylesheet Zugriff auf die Profildaten im CC/PP Profil des Requests haben. Wie dies geschieht wird im nächsten Abschnitt erläutert. Für jedes Ausgabeformat existiert ein Stylesheet. Um nun sicherzustellen, dass das richtige Stylesheet zum Einsatz kommt wird ein spezieller Selector verwendet. Dieser Selector kann Attributwerte einer Session auswerten und hierauf entsprechend verzweigen. Hierzu wird das Session Attribut `type`, das im Paket ProfileDetection aus der Konfigurationsdatei für die Profile ermittelt wurde, verwendet. Es gibt das benötigte Ausgabeformat an. Zurzeit wird HTML oder WML unterstützt.

Im letzten Schritt werden die Daten, die zwar nun das richtige Ausgabeformat besitzen, aber immer noch als SAX-Eventstream vorliegen, serialisiert und als Response an den Client gesendet. Hier wurden die Serializer für HTML bzw. WML verwendet, die Cocoon schon beinhaltet.

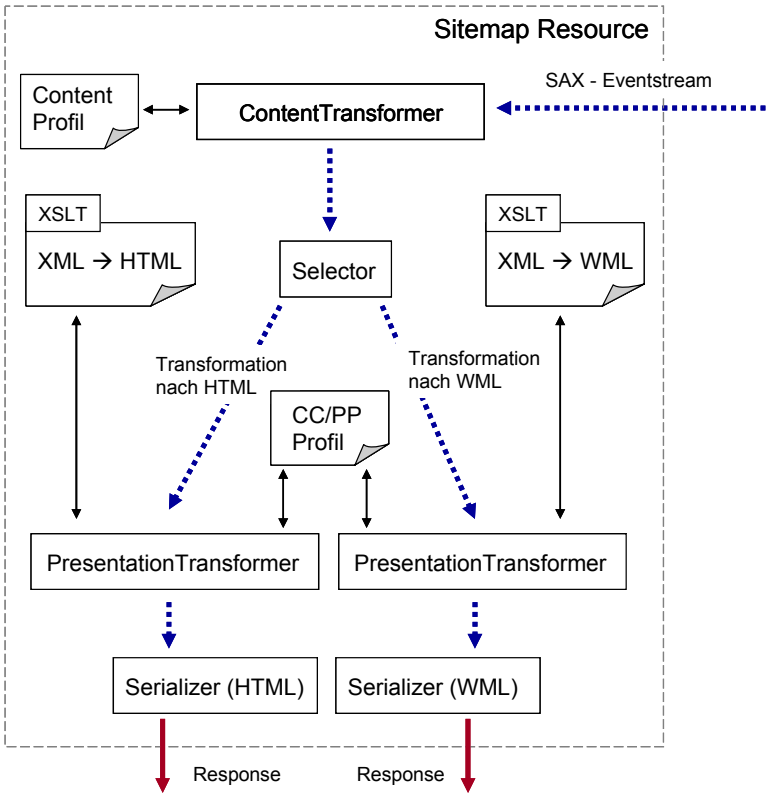


Abbildung 26: Ablauf der Customization

Nun soll noch genauer erläutert werden wie ein XSLT-Stylesheet, das vom PresentationTransformer verarbeitet wird, auf die Daten des CC/PP Profils zugreifen kann.

Hierfür musste eine Komponente entwickelt werden, die eine Verarbeitung von XSLT-Stylesheets erlaubt und einen einfachen Zugriff aus diesen Stylesheets heraus auf die Daten des CC/PP Profils ermöglicht.

Um vorhandene Funktionalitäten zu verwenden, wurde eine Komponente erstellt, die von dem in Cocoon vorhandene XSLT-Transformer erbt. Somit konnte auf einen stabilen und ausgereiften Mechanismus zur XSLT-Transformation zurückgegriffen werden. Die entwickelte Komponente erweitert die Funktion des Cocoon Transformer dahingehend,

dass nun die Daten aus dem CC/PP Profil innerhalb des XSLT-Stylesheets zur Verfügung stehen.

Im Beispiel 23 ist ein Quellcodeauszug aus dieser Komponente zu sehen. Die entscheidende Methode ist `getLogicSheetParameters`, die bei der Erzeugung der Komponente gerufen wird. Sie wird überschrieben, um einen Zugriff auf die Logiksheets zu erhalten (Logiksheets wurden im Kapitel 3.4.2 im Abschnitt XSP kurz vorgestellt).

Diese Methode ruft über `super.getLogicSheetParameters` sich selbst auf der Basisklasse, dem `TraxTransformer` auf. Hierdurch erhält sie ein Map Objekt, das alle Logiksheets im System enthält. Die CC/PP Profildaten aus der Session können nun alle in diese Map übertragen werden. Die Attribute des Profils sind die Schlüssel in der Map. Hier können zuvor auch noch die Daten aus dem Profil aufgearbeitet werden und eigenen Variablen definiert werden. Bei der Angabe der Bildschirmabmessung sind beispielsweise die Werte Höhe und Breite im CC/PP Profil als ein Attribute `screenSize` hinterlegt. Solche Daten lassen sich im Transformer in einzelne Werte extrahieren und dadurch einfacher im Stylesheet anwenden.

```
1  public class PresentationTransformer extends TraxTransformer {
2        ...
3        //Vererbte Methode des Cocoon TraxTransformer, wird bei der
4        //Initialisierung der Komponente gerufen
5        protected Map getLogicSheetParameters() {
6           ...
7           //Auslesen der Logiksheets und CC/PP Profildaten
8           Map logikSheetPara = super.getLogicSheetParameters();
9           HashMap deviceProfiles = (HashMap)
10                               session.getAttribute("deviceProfile");
11          ...
12          //Die key/value Paare des CC/PP Profils werden 1 zu 1 in
13          //die Logiksheetparameter uebertragen
14          while (iterator.hasNext()) {
15             String key = (String) iterator.next();
16             logikSheetPara.put(key,deviceProfiles.get(key));
17           }
18          ...
19       super.logicSheetParameters = logikSheetPara;
20       return super.logicSheetParameters;
21 }
```

Beispiel 23: Zugriff auf die Logiksheets eines XSLT-Stylesheets (Auszug)

Die Daten des CC/PP Profils sind nun in jedem Stylesheet verwendbar der mit diesem Transformer bearbeitet wird.
Beispiel 24 zeigt nun, wie man innerhalb eines Stylesheets die Profildaten verwendet. Zu Beginn müssen die Variablen festgelegt werden, die den Attributen des CC/PP Profils bzw. den selbst definierten Variablennamen entsprechen. In Zeile zwei und drei werden für die Bildschirmabmessung `width` und `height` definiert. Instantiiert werden müssen diese Variablen nicht, dies geschieht im Transformer mit den Werten aus dem Geräteprofil.
Die Variablen können nun an jeder beliebigen Stelle innerhalb des Stylesheets, wie in Zeile acht zu sehen, über das „$“ angesprochen werden

```
1  <!-- Definieren der Variablen -->
2  <xsl:param name="width"/>
3  <xsl:param name="height"/>
4  ...
5  <!-- Einsatz der Variablen, eine Tabelle wird auf die entsprechende
6       Bildschirmgroesse angepasst -->
7  <xsl:template match="page">
8     <table width="{$width}" height="{$height}" align="center">
9        ...
10    </table>
11 </xsl:template>
```

Beispiel 24: Verwendung der Profildaten innerhalb des XSLT-Stylesheets

5.3.4 ServiceInteraction

Dieses Paket stellt die eigentliche Anwendung dar. Hier wird das Service Konzept von M2A (vergleiche Kapitel 4.2) für den Benutzer verfügbar gemacht.
Es besteht aus fünf Transformatoren und einer zentralen Avalon Komponente, die die Interaktion mit M2A übernimmt. In Abbildung 27 ist die Anordnung der einzelnen Transformer innerhalb der ServiceSitemap zu sehen. Ein Generator in jeder Pipeline generiert aus einem XML-Dokument das Grundgerüst für den Aufbau der Seite. Abhängig von der Aktion, die ein Benutzer ausführt, wird der entsprechende Transformer benutzt. Die Transformer fügen die Daten, die das M2A-Framework liefert, in den SAX-Eventstream ein. In der Sitemap Resource Customization (Kapitel 5.3.3) werden diese Daten, bzw. der SAX-Eventstream, für den Response aufbereitet.

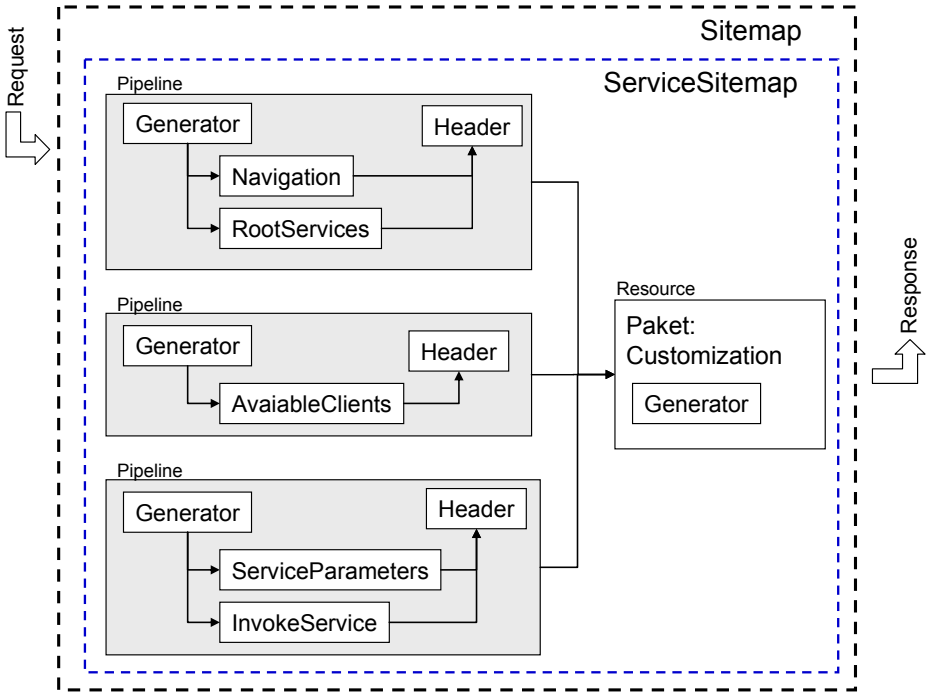


Abbildung 27: Struktur innerhalb der SubSitemap des Paketes ServiceInteraction

Bei den Transformern handelt es sich um Erweiterungen eines SAX-Transformers (`AbstractSAXTransformer`) den Cocoon bereitstellt.

Beispiel 25 zeigt das Grundgerüst eines solchen Transformers, das bei allen Transformern in diesem Paket gleich ist. Die Funktionsweise ist vergleichbar mit einem SAX-Parser (Kapitel 3.2.2). XML-Elemente im SAX-Eventstream lösen Ereignisse aus, d.h. sie rufen entsprechende Methoden im Transformer auf. Wenn ein neues Element beginnt, wird beispielsweise die Methode `startElement` gerufen. In ihr hat man nun Zugriff auf das Element, kann Attribute hinzufügen oder ein Kindelement einfügen. Man spricht hier von einem ereignisgesteuertem Verhalten. Der Transformer bemerkt, wann ein bestimmtes Ereignis auftritt (z.B. Beginn eines neuen Elements) und reagiert hierauf mit dem entsprechenden Methodenaufruf.

```
1  public class InvokeServiceTrans extends AbstractSAXTransformer {
2
3  public void setup(SourceResolver resolver, Map objectModel,
4                    String src, Parameters para) throws
5                    ProcessingException, SAXException, IOException {
6  ...
7  // Zugriff ueber den ObjectModelHelper auf den Request und die
8  // entsprechende Session
9  ...
10 }
11
12 // Diese Methode wird gerufen wenn ein neues Element im SAX-
13 // Eventstream auftaucht
14 public void startElement(String uri, String loc,
15                          String raw, Attributes attribute)
16                          throws SAXException {
17              ...
18 //Das neues Element 'service' mit einem Attribut name
19 //wird als Kind- Element hinzugefuegt
20       Attributes attribute_kind = new AttributesImpl();
21
22 attribute_kind.addAttribute(null,"name","name","RootService");
23       super.startElement(uri, "service", "service",
24 attribute_kind);
25       super.endElement(uri, "service", "service");
26                ...
27 //Element wird unveraendert in den SAX- Eventstream wieder
28 //eingefuegt
29       super.startElement(uri, loc, raw, attribute);
30 }
31
32 //Diese Methode wird gerufen wenn das Element im Sax- Eventstream
33 //wieder geschlossen wird
34 public void endElement(String uri, String loc,
35                        String raw) throws SAXException {
36                ...
37 //Element wird unveraendert in den SAX- Eventstream wieder
38 //eingefuegt
39             super.endElement(uri, loc, raw);
40 }
41 }
```

Beispiel 25: Grundgerüst eines selbst entwickelten SAX-Transformers

Im Folgenden werden die Aufgaben der Transformer aus Abbildung 27 genauer beschrieben. Jeder liefert ein Ergebnis, welches der Benutzer angezeigt bekommt. Die Abbildungen zeigen jeweils dieselben Ergebnisse, die jedoch an die verschieden Geräte angepasst wurden.

AvaiableClientsTransformer: Alle Clients (bzw. Equipments), die einem Benutzer in M2A zur Verfügung stehen, werden aufgelistet. Die Funktion eines zentralen Clienten Reposetory ist im aktuellen M2A-Prototypen noch nicht implementiert. Aus diesem Grund sind die verfügbaren Clients statisch eingetragen. Dies ist die erste Seite, die nach dem Login angezeigt wird. Der Benutzer wählt nun ein Equipment aus. Auf diesem Equipment werden mit dem folgenden Transformer alle RootServices abgefragt bzw. angezeigt.

Abbildung 28: Ansicht aller verfügbaren Clients auf einem PDA und Mobiltelefon

RootServicesTransformer: Hier werden alle RootServices eines Clients abgefragt. Der Seitenaufbau und die beschriebenen Funktionen sind auch bei der Anzeige der Services gleich, da eine RootService völlig identisch mit einem Service ist.

Wählt man einen RootService direkt an, so gelangt man eine Ebene tiefer im Service Graphen. Dies sind die Services die dieser RootService als Vorgänger besitzt. Mit [get] (siehe Abbildung 29) kann man einen RootService direkt auswählen und ausführen. Dies ist nur möglich, wenn alle Vorbedingungen hierfür erfüllt sind, also alle Vorgänger dieses

RootService einen entsprechenden Zustand besitzen. Ist dies nicht der Fall, ist [get] ausgegraut. Besitzt ein RootService keinen Vorgänger, so ist er selbst ausgegraut. Wurde ein RootService bzw. Service schon ausgeführt, wird er entsprechend gekennzeichnet. Einen RootService kann man beliebig oft ausführen. Jeder RootService kann weiterhin eine Beschreibung besitzen, die hier ebenfalls angezeigt werden kann.
Alle beschriebenen Funktionalitäten gelten ebenso für die Anzeige der Services, da ein RootService identisch mit einem Service ist und nur einen Einstiegspunkt auf einem Equipment darstellt.

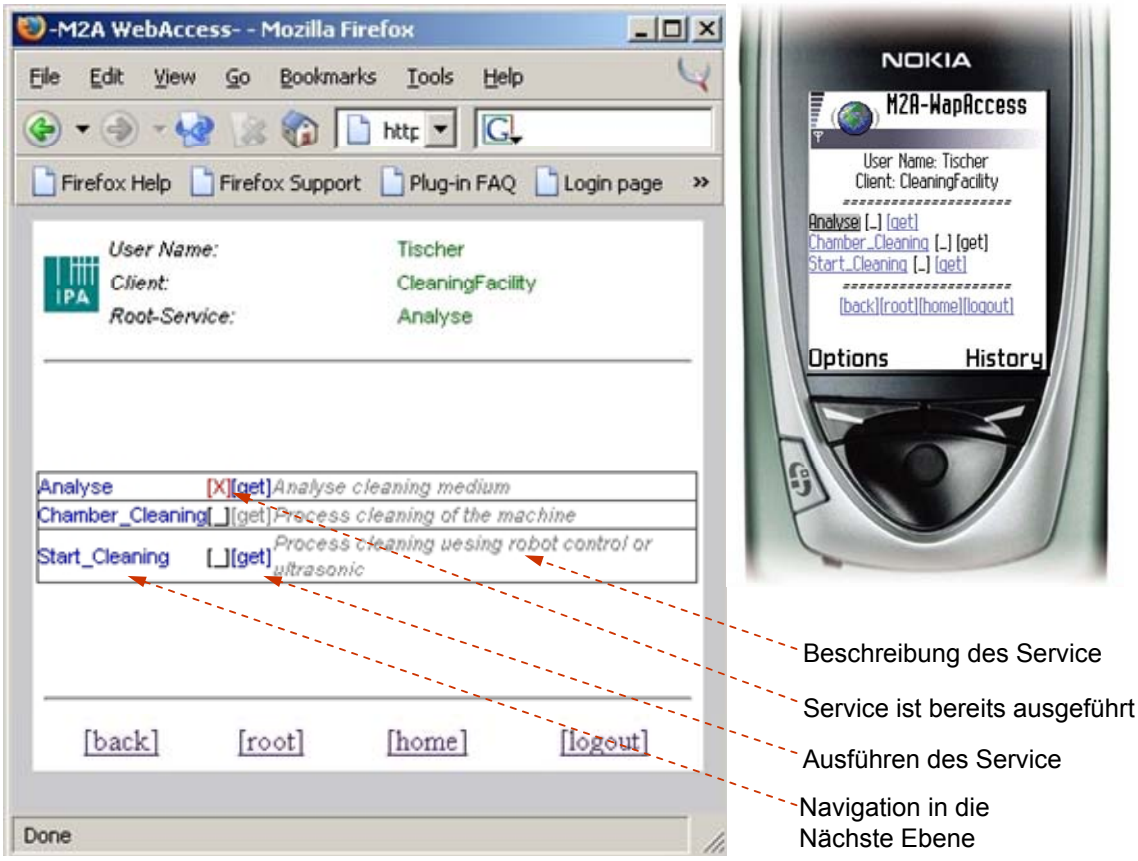


Abbildung 29: Seite mit allen RootServices im System

NavigationTransformer: Dieser Transformer erlaubt eine Navigation durch den Service-Graphen auf einem Client. Es ist eine vor- und zurück Navigation (Ebene tiefer bzw. Ebene höher) möglich. Die Services auf jeweiligen Ebenen werden dem Benutzer dargestellt.

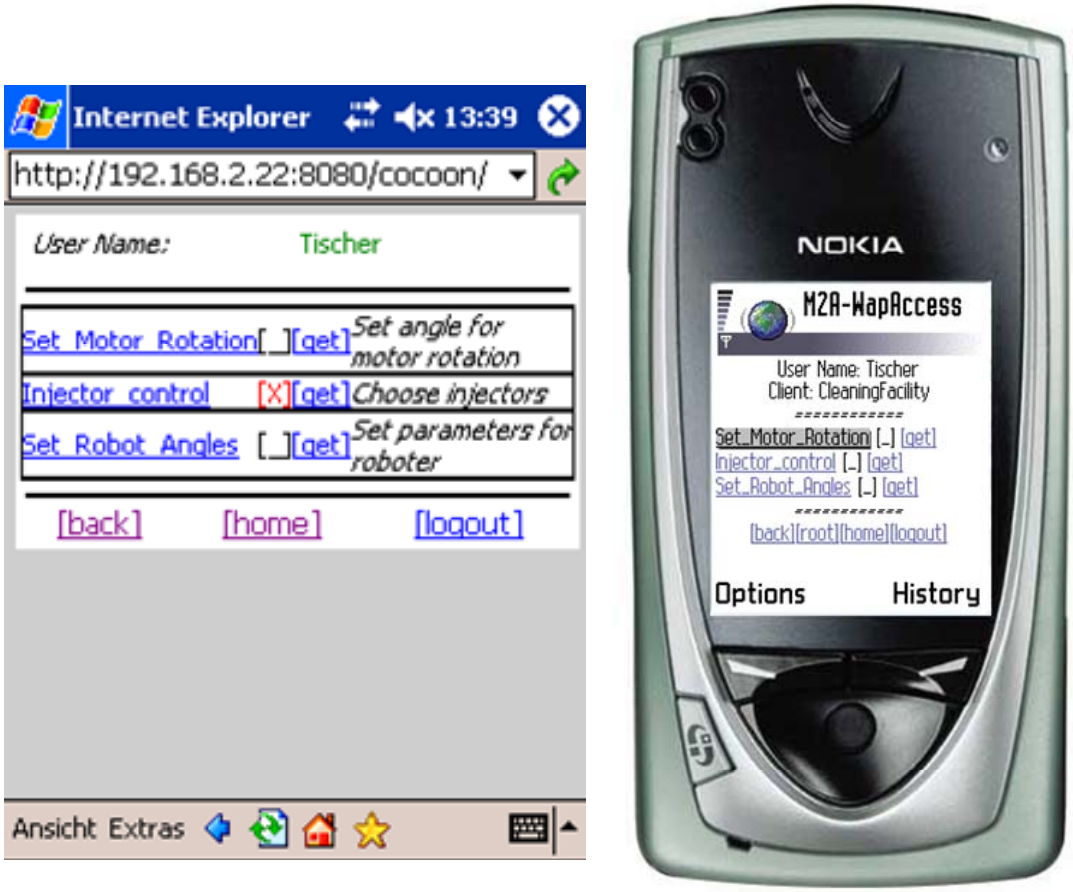


Abbildung 30: Anzeige aller Services auf einer Ebene in dem Service Graphen

ServiceParametersTransformer: Ein Service der mit [get] ausgewählt wurde, muss auf seine benötigten Übergabeparameter untersucht werden. Hieraus wird eine entsprechende Eingabemaske generiert, in die der Benutzer die benötigten Daten einträgt. Über [invoke] wird der ausgewählte Service mit den eingegebenen Parametern ausgeführt. Dies geschieht im folgenden Transformer, dem InvokeServiceTransformer.

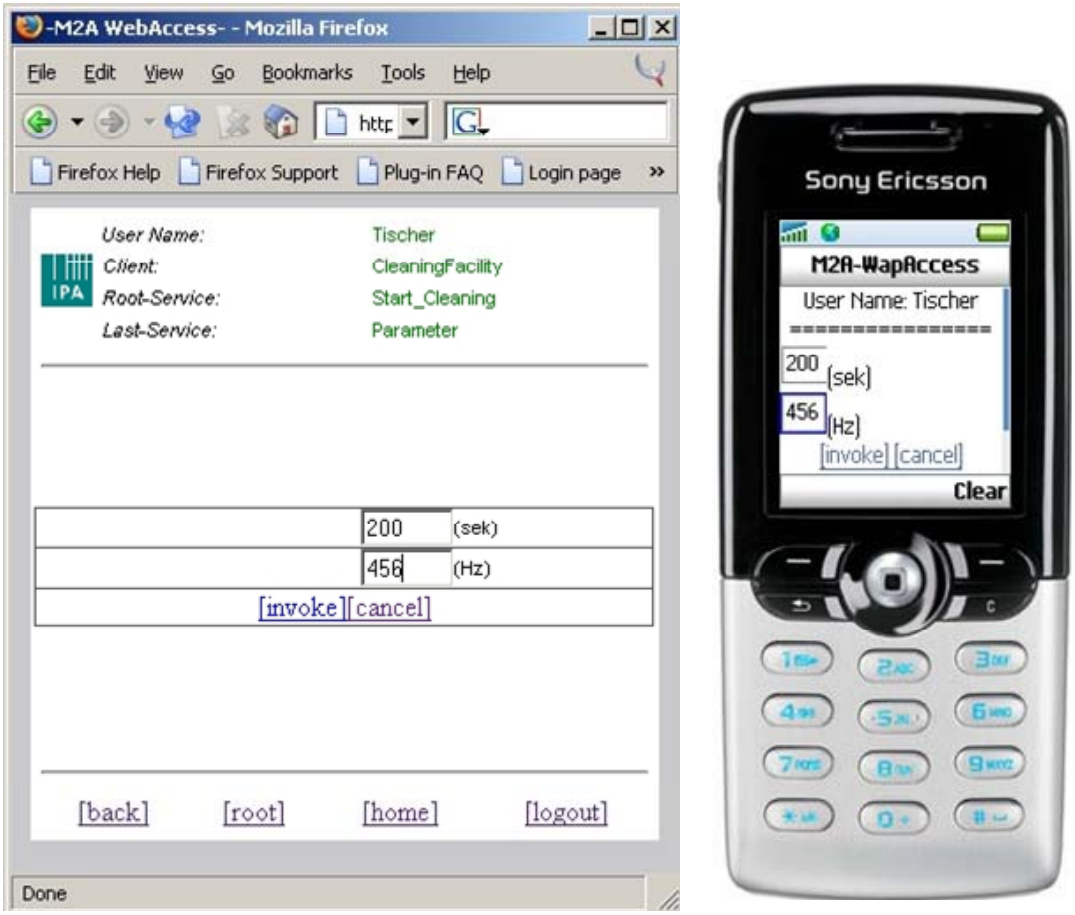


Abbildung 31: Eingabemaske der Parameter für das Ausführen eines Service

InvokeServiceTransformer: Hier wird letztlich ein Service ausgeführt. Zuerst werden die Parameter, die der Benutzer zuvor für diesen Service eingeben hat, ausgelesen und auf ihren richtigen Datentyp überprüft. Im zweiten Schritt wird mit den Parametern der Service letztlich ausgeführt. Das Ergebnis des Service Aufrufs wird angezeigt. Mit [ok] gelangt der Benutzer wieder in die Ebene des Service Graphen, von wo er den Service aufgerufen hat.

Abbildung 32: Bestätigungsseite eines ausgeführten Service

HeaderTransformer: Im Anschluss an jeden der vorherigen Transformer folgt in der entsprechenden Pipeline noch der HeaderTransformer. Er fügt alle Informationen, die im Header der Seite zu sehen sind hinzu. Es kann hier der Benutzername, Name des ausgewählten Equipments und RootService sowie der zuletzt ausgeführte Service angezeigt werden.

In Abbildung 33 sieht man das Klassendiagramm des Paketes. Die Schnittstelle zum M2A-Framework stellt das Interface `ServiceInteraction` da. Es bietet alle Zugriffsmöglichkeiten auf ein Equipment bzw. Service, die im Interface `ServiceAccess` das im Kapitel 4.2 über das Service Konzept von M2A beschrieben wurde. `ServiceInteraction` bzw. die konkrete Implementierung `ServicInteractionImpl` sind nach dem Avalon Komponentenmodell entwickelt worden. Für alle Transformer fungiert diese Komponente als Datenquelle. Über den Service Manager können sie eine Instanz dieser Komponente holen und benutzen.

`ServiceInteraction` hat Zugriff auf die Session des HTTP-Request und somit auf den M2A-Adapter und die entsprechende M2A-Session. Hierüber kann sie mit dem M2A-Framework bzw. mit den daran angeschlossenen Equipments kommunizieren. Eine weitere Aufgabe ist es den Navigationspfad des Benutzers mitzuverfolgen und in der Session zu speichern. Ohne diesen wäre keine Rücknavigation im Service-Graphen möglich.
In der statischen Klasse `XMLElements` sind alle Elemente- und Attributname, die von den Transformatoren benötigt werden um den SAX-Eventstream zu beeinflussen, als Variablen definiert.

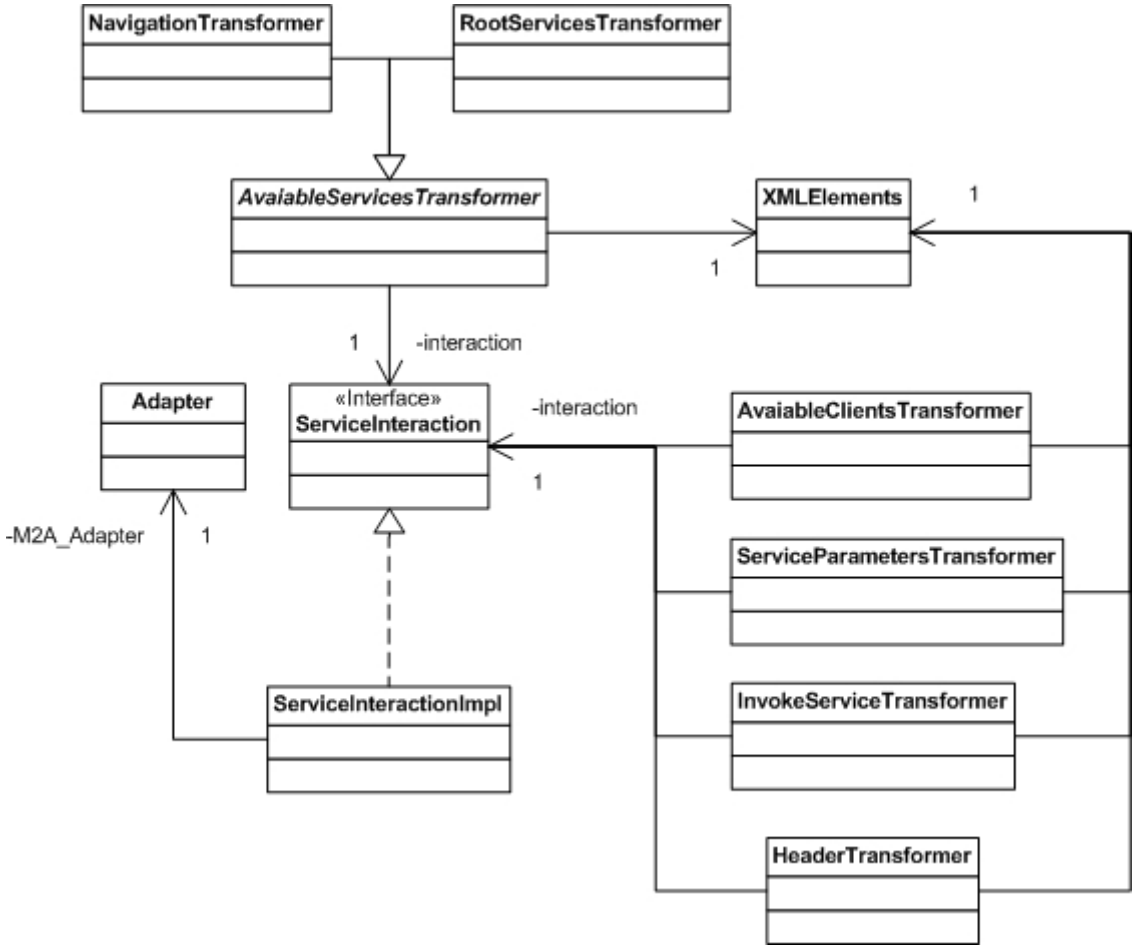


Abbildung 33: Klassendiagramm des Paketes ServiceInteraction

5.3.5 Logging

Für das Logging wurde das Unterprojekt LogKit von Avalon, eingesetzt. Auch das Logging innerhalb von Cocoon beruht hierauf, und deswegen wurde es auch für das Logging in dieser Arbeit ausgewählt.
Um nun einer Komponente das Logging zu ermöglichen, muss sie das entsprechende Interface implementieren. Vom Avalon Service Manager erhält die Komponente bei der Initialisierung ein Objekt vom Typ Logger, das die entsprechenden Methoden für das Logging bereitstellt. Das Logging kann in fünf verschiedenen Stufen, den so genannten Log-Levels, erfolgen:

1. DEBUG – Allgemeines Debugging, beispielsweise während der Entwicklung.
2. INFO – Informationen über den allgemeinen Programmverlauf.
3. WARN – Ein Fehler ist aufgetreten, der allerdings zu keinem Abbruch der Verarbeitung führte.
4. ERROR – Der aufgetretene Fehler führt dazu, dass das System nicht mehr oder nur noch teilweise korrekt arbeitet.
5. FATAL_ERROR – Ein schwerwiegendes Problem ist aufgetreten. Das System ist nicht mehr in der Lage weiter zu arbeiten.

Es ist nötig den Logger, den die Komponente vom ServiceManager zugewiesen bekommt, in einer Logging- Konfigurationsdatei zu definieren. Hier sind alle Logger, die Cocoon benutzt, eingetragen. Auch die Angabe des Log Levels erfolgt hier. Ein Logger, der WARN als Level zugewiesen bekommen hat, wird auch alle Meldungen der niedrigeren Levels aufzeichnen. Dies sind ERROR und FATAL_ERROR.
Das Zuweisen eines Loggers zur Komponente erfolgt in der zentralen Konfigurationsdatei von Cocoon. Ein Auszug hieraus ist in Kapitel 3.5.2 , Beispiel 14, zu sehen. Jede Komponente kann ihren eignen Logger zugewiesen bekommen oder mehrere Komponenten können auf einen gemeinsamen Logger zugreifen.

Beispiel 26 zeigt einen Ausschnitt aus der Komponente `ProfilesReader`, in der die Profildaten erfasst werden (5.3.1 ProfileDetection). Hier wurde von der abstrakten Klasse `AbstractLogEnabled` geerbt, die vom Avalon Framework bereitgestellt wird und die Integration des Logging vereinfacht. Den Zugriff auf das entsprechende Objekt erhält man

nun einfach über die Methode `getLogger`. Aus Performancegründen ist es immer ratsam abzufragen, ob der entsprechende Log-Level auch gesetzt ist. Jeder Level besitzt eine entsprechende Methode, um dies festzustellen. Im Beispiel wird vor dem Loggen auf dem Level INFO mit `isInfoEnable` abgefragt (Zeile fünf), ob dieser Level bzw. ein höherer, auch aktiviert ist. Ist dies der Fall, wird die Meldung in den Logger geschrieben.

```
1 public class ProfilesReaderImpl extends AbstractLogEnabled
2                                 implements  ProfilesReader {
3  ...
4 // Die Meldung wird in den Logger geschrieben
5  if (getLogger().isInfoEnabled())
6       getLogger().info("Found matching profiledata for
7                         user-agent: " + this.userAgent);
8  ...
9 }
```

Beispiel 26: Das Logging in der Komponente ProfilesReader

5.3.6 Exception Handling

Das Exception Handling erfolgt zum ersten in der jeweiligen Komponente. Die aufgetretene Exception wird abgefangen, eine entsprechende Meldung in den Logger geschrieben und der StackTrace auf der Konsole ausgegeben. Erlaubt der Fehler ein normales Weiterarbeiten mit dem System, geschieht weiter nichts. Ist der Fehler schwerwiegender, muss der Benutzer über das Auftreten dieser Exception bzw. den Fehlergrund informiert werden.

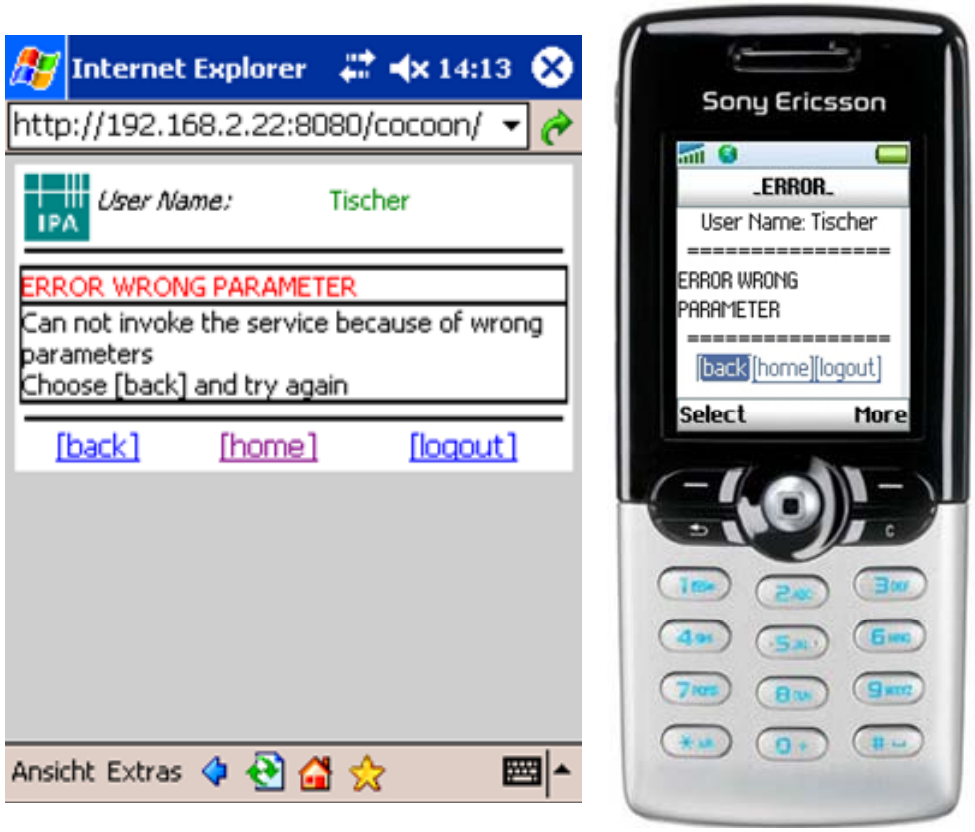


Abbildung 34: Fehler bei falscher Eingabe eines Parameters

Zur Generierung von entsprechenden Fehlerseiten, in Abhängigkeit der aufgetretenen Exception, bietet Cocoon einen entsprechenden Mechanismus. Eine Exception, die von einer Komponente innerhalb einer Pipeline verursacht, und nicht intern verarbeitet wird, führt zu deren Abbruch. Nun kann eine Fehlerbehandlung innerhalb der Pipeline erfolgen. Hierfür sind die `<map:handle-errors>` Abschnitte zuständig. Die Exception wird in diesem Abschnitt gefangen und kann nun weiter verarbeitet werden. Dies ist vergleichbar mit dem `catch` Block in Java.

Beispiel 27 zeigt die Fehlerbehandlung bei falscher Parametereingabe vor dem Ausführen eines Services. Die Exception wird von der Komponente `invokeServiceTransformer` verursacht. Die Fehlerbehandlung der Pipeline fängt sie ab. Hier wird ein spezieller Selector eingesetzt, der so konfiguriert werden kann, dass in Abhängigkeit der aufgetretenen Exception verschiedene Aktionen ausgeführt werden. Im Beispiel wird anhand der aufgetretenen `NumberFormatException` eine XML-Datei eingelesen, die eine individuelle Fehlerbeschreibung für den Benutzer enthält. Anschließend wird die Resource `errorTransformation` aufgerufen, die die Fehlermeldung für das jeweilige Endgerät

aufbereitet. Diese Resource arbeitet genau, wie im Paket Customization, Kapitel 5.3.5, beschrieben. Der einzige Unterschied besteht darin, dass andere XSLT-Stylesheets zur Aufbereitung der Daten für das nötige Ausgabeformat eingesetzt werden.

```
<map:pipeline>
      <map:match pattern="SERVICE" type="wildcard">
            ...
        <!—Auftreten einer NmberWrongFormatException -->
              <map:transform type="invokeServiceTransformer"/>
            ...
           </map:match>

         <!-- Verarbeitung der Exception am Ende der Pipeline -->
          <map:handle-errors>

         <!--Fangen der Exceptions -->
         <map:select type="exception">
            <map:when test="numberFormatException">
               <map:generate src="xml/error/wrongInputFormat.xml"/>
               <map:call resource="errorTransformation"/>
            </map:when>
            ...
         <!-- Beliebig weiter Exception koennen abgefangen
              werden -->
         </map:select>
         </map:handle-errors>
</map:pipeline>
```

Beispiel 27: Fehlerbehandlung bei falscher Parametereingabe

6 Ausblick

Die Möglichkeiten der Steuerung und Überwachung von Anlagen über Mobile Endgeräte ist ein relativ junges Themengebiet, das vielfältige Möglichkeiten eröffnet. Speziell in der Halbleiterindustrie können durch die Steuerung der Anlagen außerhalb einer Reinstraumumgebung die aufwendigen Umkleideprozeduren für die Mitarbeiter verringert werden und einer Verunreinigung des Reinraumes vorgebeugt werden.
Über zwei Nachteile der Ansteuerung über einen Webserver muss man sich hierbei allerdings im Klaren sein. Zum einen kann das HTTP-Protokoll nicht gewährleisten, dass eine Verbindung zum Server auch wirklich besteht bzw. der Benutzer wird es nicht bemerken, wenn diese Verbindung unterbrochen ist.
Auch ist eine Benachrichtigung über eine Zustandsänderung auf einer Maschine ohne eine zusätzliche Software auf dem Clientgerät nicht möglich. Da es keinen echten HTTP-Push Mechanismus gibt, der die Aktualität der Daten, insbesondere bei zeitkritischen Anwendungen, gewährleistet.
Diese beiden Tatsachen liegen in der Natur des HTTP-Protokolls, zeigen jedoch, dass von einer sinnvollen Steuerung kritischer Abläufe auf Maschinen über den Webserver abzuraten ist. Eine auf das jeweilige Endgerät zugeschnittene Softwarelösung (M2A-MobileAdapter) ist hier unumgänglich.
Die Anwendungsgebiete für die Steuerung über den Webserver sollten auf unkritische Bereiche, zur Abfrage von Prozessdaten oder der Fernwartung beschränkt werden.
Der Vorteil einer rein Webserver basierten Lösung ist sicherlich, dass keine zusätzliche Software auf den Geräten installiert werden muss und die Anzeigemöglichkeiten der Informationen lassen sich in den Browsern sehr flexibel gestalten.

Die Service Struktur von M2A ist für einen produktiven Einsatz noch nicht ausreichend, da sie von Anfang an nur für Demonstrationszwecke vorgesehen war. Hier muss noch eine Nachbesserung erfolgen. Gegebenenfalls muss sie komplett neu erstellt werden. Dies zeigte sich klar bei der Integration der 3D Reinigungsanlage. Eine Abbildung aller Services war zwar möglich, allerdings ist der Implementationsaufwand noch zu groß. Für den Einsatz in einem Produktionsumfeld bedeutet dies einen zu großen Zeit- und Kostenaufwand.

Des Weiteren währe eine Grafische Oberfläche sinnvoll, die den Anwender bei der Integration von Anlagen unterstützt. Ein erstellter Service könnte hier komfortabel via „drag and drop“ in die Service Struktur einer Anlage übernommen werden.

Die CC/PP Profile stellen eine ausgereifte Möglichkeit zur Beschreibung von Endgeräten zur Verfügung und mit dem W3C Konsortium im Rücken hat es eine gute Chance, sich als Standard auf diesem Gebiet zu etablieren. Die Informationen aus den Profilen lassen sich für die Adaption der Daten auf dem Webserver noch weiter ausschöpfen. Weiterhin ist zu überlegen, ob die CC/PP Profile auch innerhalb des M2A-Frameworks zum Einsatz kommen. Mit ihnen könnten Entscheidungen über das Routing von Nachrichten zu verschieden Clienten getroffen werden.

Die Einarbeitung in das Cocoon Framework war zu Beginn recht schwierig. Das Pipeline Konzept ist anfangs etwas unübersichtlich. Auch bei der Erstellung von eigenen Komponenten ist sehr viel Hintergrundwissen über die Architektur von Cocoon und dem Prinzip der komponentenbasierten Softwareentwicklung mit Avalon nötig.
Hat man allerdings einmal diese Hürden überwunden, steht einem ein äußerst mächtiges Werkzeug zur Verfügung, das sehr viele Möglichkeiten eröffnet und komfortable Funktionen zur Verfügung stellt. Zur Verwirklichung kleiner Projekte ist die Einarbeitung in Cocoon allerdings zu aufwendig.

Das Bereitstellen und Verarbeiten der Eingabemasken für die Eingabe der Parameterdaten zur Ausführung der Services kann im Webserver noch verbessert werden. Hier bietet Cocoon die Unterstützung von Formular-Frameworks wie JXForms oder dessen Weiterentwicklung Woody.

7 Anhang

7.1 Abkürzungsverzeichnis

API	Application Programming Interface
ASP	Active Server Pages
BMWA	Bundesministerium für Wirtschaft und Arbeit
CC/PP	Composite Capabilities/Preferences Profile
DTD	Document Type Definition
DOM	Document Object Model
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAS	Institut für Software und Automatisierungstechnik
IoC	Inversion of Control
IPA	Institut für Produktion und Automatisierung
JSP	Java Server Pages
M2A	Message to Anywhere
MDA	Mobile Digital Assistant
MIME	Multipurpose Internet Mail Extension
MVC	Model View Control
OMA	Open Mobile Allinace
PDA	Personal Digital Assistant
PDF	Portable Document Format
PHP	Hypertext Preprocessor
RAM	Random Access Memory
RDF	Resource Description Framework
SAX	Simple API for XML
SDK	Software Development Kit
SPS	Speicherprogrammierbare Steuerung
SVG	Scalable Vector Graphics

SoC	Separation of Concerns
UAProf	User Agent Profile
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAP	Wireless Access Protocol
WML	Wireless Application Protocol
XML	Extensible Markup Language
XPath	XML Path Language
XSLT	Extensible Stylesheet Language Transformation
XSP	Extensible Server Pages

7.2 Literaturverzeichnis

[Apache04]
Apache Software Foundation: Die Apache Software Foundation; 2004.
URL: http://www.apache.org

[ApacheXML04]
Apache Software Foundation: Das Apache XML Projekt; 2004.
URL: http://xml.apache.org

[Avalon04]
Apache Software Foundation: Das Apache Avalon Framework; 2004.
URL: http://avalon.apache.org

[Brogden03]
Brogden, Bill; D'Cruz; Gaither Mark: Cocoon 2 Programming- Web Publishing with XML and Java; San Francisco, London: SYBEX, 2003.

[CCPP04]
W3C: Structure and Vocabulary: Composite Capability/Preferences Profiles; 2004.
URL: http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115

[CTspecial04]
c't special: Zusammenfassung der Tabellen auf Seite 17-19, 40-41, 50-53 und 66-67; Hannover: Heise Zeitschriften Verlag, c't spezial, 2004.

[Cocoon04]
Apache Software Foundation: Das Cocoon Webframework; 2004.
URL: http://cocoon.apache.org

[CocoonAPI04]
Apache Software Foundation: Cocoon API 2.1.5 Dokumentation; 2004.
URL: http://cocoon.apache.org/2.1/apidocs/index.html

[CocoonDoc04]
Apache Software Foundation: Cocoon Benutzer Dokumentation; 2004.
URL: http://cocoon.apache.org/2.1/userdocs

[DOM02]
W3C: Document Object Model; 2002.
URL: http://www.w3.org/DOM

[Geese01]
Geese, Elmar; Heiliger, Markus; Loher Matthias: XML, XSLT, VB und ASP; Bonn: Galileo Press, 2001.

[Gerlicher04]
Gerlicher, Ansgar; Rupp Stephan: Symbian OS, Eine Einführung in die Anwendungsentwicklung; Heidelberg: dpunkt, 2004.

[Jaboring04]
Jaboring, Johannes: Comparison of Environment/Capabilities/Profiles; Klagenfurt: Semesterarbeit Universität Klagenfurt, 2004.

[JBoss04]
JBoss Inc.: Der JBoss Open Source Applicationserver; 2004.
URL: http://www.jboss.org

[Kumar04]
Kumar, Pankaj; Srinivas, Davanum: Understanding Apache Cocoon; 2004
URL: http://cocoon.apache.org/2.1/userdocs/concepts

[Langham02]
Langham, Matthew: Die Verpuppung;
Frankfurt: Software und Support Verlag, Java Magazin Nr. 3, 2002.

[M2A04]
Fraunhofer IPA: Das Message to Anywhere Frameworks; 2004.
URL: http://www.mtoa.de

[Niedermeier03]
Niedermeier, Stephan: Cocoon Tutorial; 2003.
URL: http://www.logabit.com/cocontutorial

[Niedermeier04]
Niedermeier, Stephan: Cocoon 2 und Tomcat; Bonn: Galileo Computing, 2004.

[Oestereich04]
Oestereich, Bernd: Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language; München, Wien: Oldenburg, 2004.

[OpenMobileAllinace02]
Open Mobile Allinace Ldt; 2002.
URL: http://www.openmobilealliance.org

[RDFRec04]
W3C: Resource Description Framework (RDF): Concepts and Abstract Syntax, Recommendation; 2004.
URL: http://www.w3corg/TR/rdf-concepts

[RRZN01]
Regionales Rechenzentrum für Niedersachsen: XML 1.0-Grundlagen; Hannover, 2001.

[SAX04]
SourceForge Project: Simple API for XML (SAX); 2004
URL: http://sax.sourceforge.net

[Saxon04]
SourceForge Project: Der SAXON XSLT und XQuery Processor; 2004.
URL: http://saxon.sourceforge.net

[Seeboerger02]
Seeboerger - Weichselbaum, Michael: Java/XML; Bonn: bhv,2002.

[Struts04]
Apache Software Foundation: Das Struts Webframework; 2004.
URL: http://struts.apache.org

[Tomcat04]
Apache Software Foundation: Der Tomcat Webserver; 2004.
URL: http://jakarta.apache.org/tomcat/

[Turbine04]
Apache Software Foundation: Das Turbine Webframework; 2004.
URL: http.//Jakarta.apache.org/turbine/

[UAProfReposetory04]
Feher, Pal Tamas: UAProf Profile (UAProf) Reposetory; 2004.
URL: http://w3development.de/rdf/uaprof_repository

[Velocity04]
Apache Software Foundation: Das Templating Framework Velocity; 2004.
URL: http://Jakarta.apache.org/velocity/

[W3C04]
World Wide Web Consortium (W3C); 2004.
URL: http://w3c.org

[WAGUAProf01]
WAG UAProf, Wireless Application Protocol; 2001.
URL: http://www.wmlclub.com/docs/especwap2.0/WAP-248-UAProf-20010530-p.pdf

[Wang04]
Wang, Dapeng: Bunte Rahmen, Überblick über Webframeworks;
Frankfurt: Software und Support Verlag, Java Magazin Nr. 9, 2004.

[Wafer04]
Web Application Framework Research Project,
URL: http://www.waferproject.org

[XmlRec00]
W3C: Extensible Markup Language (XML) Recommendation; 2000.
URL: http://www.w3.org/TR/2000/REC-xml-20001006

[XPathRec99]
W3C: XML Path Language (XPath) Recommendation; 1999.
URL: http://www.w3c.org/TR/xpath

[XSLRec01]
W3C: Extensible Stylesheet Language (XSL) Recommendation; 2001.
URL: http://www.w3c.org/TR/xsl

[Ziegler03]
Ziegler, Carsten: Der Nebel von Avalon;
Frankfurt: Software und Support Verlag, Java Magazin Nr. 1, 2003.